INSTITUTE OF SIMULATION SCIENCES
DEPARTMENT OF MATHEMATICAL SCIENCES
FACULTY OF COMPUTING SCIENCES AND ENGINEERING
DE MONTFORT UNIVERSITY

SOFT COMPUTING AND FRACTAL GEOMETRY IN SIGNAL PROCESSING AND PATTERN RECOGNITION

JUNE 2000

ARDESHIR OSANLOU CMath, MIMA

PROFESSOR JOHNATHAN M. BLACKLEDGE
DR BRIAN BRAMER

ACKNOWLEDGEMENTS

I wish to express my gratitude to Professor J. M. Blackledge (Director of Studies) for giving me the opportunity to work with him in this field, his valuable and constructive guidance, and his full support in all aspects of this project. I particularly appreciate the induction course, which allowed me attending all the modules on the advanced MSc course in Digital Signal and Image Processing.

My sincere thank to Dr Brian Bramer (Second Supervisor), to whom my association goes back to 1989, when I was jointly covering the practical sessions of his C/Assembly language programming on an MSc IT course. His active involvement with the students wet my appetite to become a qualified F. E. teacher.

My thanks to the Institute of Simulation Sciences, the Department of Mathematics, and De Montfort University Library.

Finally, I express my appreciation to my wife Valerie, my sons Orod and Rostam, for their full support, and to my parents Amin and Amineh for their unselfish support of my personal development. This work is dedicated to them all.

ABSTRACT

This research encompasses a comprehensive study and analysis of adaptive computing techniques and fractal geometry in digital signal processing and pattern recognition. After starting with an introduction to the subject areas, links between artificial neural networks and fractal geometry are covered, followed by practical work. Using a pattern recognition problem, the whole process of applying artificial neural networks to a bio-chemical application has been practically analysed. By developing a robust weight set their robustness to noisy data were examined.

Since the 1990s digital signal communication has been the focus of considerable attention, resulting in a rapidly developing field. Using a digital communication system designed for secure data transmission, adaptive neuro-models were developed to optimise the system's performance in the presence of various types of noise, overcoming the system's error rates generated with low signal/noise ratios (SNRs).

The most significant finding was that random noise improves model performance in terms of generalisation abilities, making adaptive neuro-models particularly valuable in dealing with transmission noise, and unwanted signals in digital communication systems.

CONTENTS

NOTATION AND GLOSSARY

ANN	Artificial Neural Networks
BP	Back-propagation
DC	Direct Current.
DFT	Discrete Fourier Transform
FBm	Fractional Brownian motion
FDS	Fractional Dimension Segmentation.
FT	Fourier Transform
FFT	Fast Fourier Transform
IFS	Iterated Function System.
PDF	Probability Density Function.
PSDF	Power Spectrum Distribution Function.
RSF	Random Scaling Fractal
SFD	Stochastic Fractional Differentiation
SNR	Signal to Noise Ratio
$\otimes$	Convolution function $f \otimes g = \int_{-\infty}^{\infty} f(x)g(x'-x)dx$
$\odot$	Correlation $f \odot g = \int_{-\infty}^{\infty} f(x)g(x-x')dx$
$\langle\cdot\rangle$	Expected or mean value of …
$\lfloor\cdot\rfloor$	Value … rounded down.
$\inf\{\cdot\}$	The infimum or minimum
$\sup\{\cdot\}$	The supremum or minimum
σ, σ^2	Standard Deviation, Variance.
$\Gamma(x)$	The Gamma Function $= \int_0^{\infty} t^{x-1}e^{-t}dt$.

$\hat{F}\{\cdot\}, \hat{F}^{-1}\{\cdot\}$	Forward and Inverse Fourier Transform
$Im(\cdot)$	Imaginary part of a Complex Number.
$Pr[\cdot]$	Probability Density Function of …
$Re(\cdot)$	Real part of a Complex Number.
$\mathrm{Sinc}(x)$	$\sin(x)/x$.
D	Fractal Dimension
D_B	Box-counting Dimension
D_C	Correlation Dimension
D_F	Fourier Dimension
D_H	Hausdorff-Besicovitch Dimension .
D_P	Pointwise Dimension
D_s	Similarity dimension
$P(k)$	Fourier Power Spectra
$\hat{P}(k)$	Least Squared Fit to $P(k)$.

1 INTRODUCTION

1.1 FRACTAL GEOMETRY

The geometric interpretations of fractional partial differential equations can be explored in terms of fractal geometry. Fractal geometry is suitable for describing natural objects. It deals with shapes of infinite detail and allows defining and measuring the properties of such objects (Mandelbrot, 1983). This measure is compounded in a metric called the 'Fractal Dimension' or the 'Similarity Dimension'.

The concept of self-similarity is central to fractal geometry. It means that some types of mainly naturally occurring objects look similar at different scales. There are two distinct types of fractals, which exhibit this property: deterministic fractals, which are objects that look identical at all scales, and random fractals, which do not generally posses such deterministic self-similarity.

Dynamical systems theory provides the framework for the generation of fractals as well as discussing chaos (Weeks and Burgess, 1997). This is an important point to highlight in regard of ANNs as the dynamics of all types of neural network can be analysed and explained by using the fixed point theory in the traditional discrete dynamical system (Rabinovich, 1996).

1.2 SOFT COMPUTING

Soft computing methodologies form the basis of an emerging approach to constructing computationally intelligent systems. The term "soft computing" encompasses Artificial Neural Networks (ANNs), Fuzzy Logic (FL), and Probabilistic Reasoning (PR) with the latter subsuming simulated annealing, generic algorithms and chaotic systems.

Professor (Lotfi Zadeh, 1992), who introduced the concept of fuzzy sets in 1965 in a publication – *Fuzzy Sets, Information and Control, 8, 338-353*, defines soft computing as follows:

'Soft computing is an emerging approach to computing in an environment of uncertainty and imprecision. The essence of soft computing is that unlike the traditional, hard computing,

soft computing is aimed at an accommodation with the pervasive imprecision of the real world. Thus, the guiding principle of soft computing is to exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, low solution cost, and better rapport with reality'.

Soft computing techniques may be applied in real world applications where conventional computing systems are currently unable to cope, e.g. adaptability to or immunity to noise. The role model for soft computing is the mammalian brain, with the artificial neural networks providing a central contribution in learning and adaptation.

ANNs can be described as parallel distributed information-processing structures made up of processing elements; the elements being simplified versions of biological neurones. Because they are not based on rigid algorithms, neural networks are naturally fault tolerant. They can process many inputs and produce many outputs, hence they are applicable to multivariable systems and potentially offer fast processing.

The strength of fuzzy logic is knowledge representation using fuzzy if-then rules, and that of simulated annealing and generic algorithms is systematic random search and optimisation.

1.3 SOFT COMPUTING IN SIGNAL PROCESSING AND PATTERN RECOGNITION

Since the 1970's, many mathematically optimal solutions have been developed for specific tasks in signal processing and identification of objects from observed patterns or images. These solutions break down however, when adapted for similar problems or when the underlying assumptions that may be unreasonable in practice are violated. For example, the common assumption that noise follows a particular distribution is known to play a detrimental role in many experimental situations. As a result, engineers and scientists active in signal processing and pattern recognition have been showing a growing interest in soft computing methodologies. Numerous publications, conference sessions, as well as many applications are indicative of this expanding process.

Unlike the traditional approaches, artificial neural networks in partnership with complementary methodologies, which form the basis of soft computing, can tolerate the imprecision and uncertainty of the real world. They have been used successfully for many

recognition and classification, chaotic time series prediction, texture recognition and image understanding types of problems (Caudill and Butler, 1992).

1.4 AIMS AND OBJECTIVES

The principal aims of this research were to investigate and practically analyse the suitability of neural networks and fractal geometry for digital signal communication, in particular in the areas of neuro-modelling and filtering of unwanted signals. To achieve this the following objectives were set:

A review of digital signal processing techniques, fractal geometry, and the fundamental concepts of neural network technology.

An investigation of any links between artificial neural networks and fractal geometry, and suitability of neuro-fractal models for noisy pattern recognition and signal processing tasks.

Using pattern recognition problems to address issues regarding noise, incorporation of approximate models, mapping, and generalisation attributes of suitable paradigms.

Developing a robust neuro-model for a digital communication system that employs fractal modulation for secure transmission, to:

- overcome the system's error rates generated with low signal/noise ratios, and
- optimise the decoding process in terms of speed, and robustness of technique in presence of various noise fields.

In particular it was considered that the benchmark experiments would provide a number of main contributions to this work. These included: an analysis of neural networks robustness to noisy data that occurs in industrial applications (employing fractal noise convolution), a study of the practical issues involved in the development of static/dynamic models (e.g. the stability problem), and an analysis of the issues concerned with the quality, representativeness and pre-processing of the training/testing data.

1.5 THESIS OVERVIEW AND OUTLINE

1.5.1 Overview

In 1996, when I started my PhD, I was already familiar with artificial neural networks (ANNs) and their potential of being used for many real world applications in a noisy environment. As a consultant at Fisher Controls International, Leicester, in the previous year, I had carried out suitability studies of ANNs in process control and published two internal papers. The material included in Appendix B, which covers preliminary unifying information on ANNs, form part of one of the publications which was concerned with ANNs in process control. The second paper, looked at ANNs from a practitioners viewpoint, categorising them into either static or dynamic networks.

In 1997/8, after completing an induction course in digital signal processing and fractals, I carried out work on fractal modulation and looked for any links between ANNs and fractal geometry. At the same time, I carried out practical work on a benchmark process control problem, addressing issues regarding noise, and incorporation of approximate models. The practical work indicated robustness of neuro models to noisy data, both in term of training and generalisation abilities.

The question, then became how such robustness to noisy data can be used to solve problems where conventional solutions have proved inadequate, if not impossible. As a result, I then considered developing neuro-models using data with the following transformation:

- Fourier type transform – (Data compression)
- Wavelet transform – (Multi-resolution analysis)
- Fractal transform – (Self similarity of input pattern)

Utilising my practical findings on robust neuro models and using fractal analysis, based on

$$\frac{d^{\,q(t)}}{d\,t^{\,q(t)}} f(t) = n(t)$$

n (t) – White noise

f (t) - Fractal time series

q(t) - Fractal dimension signature

$$1 < q(t) < 2 \qquad \forall t$$

models were developed for a digital signal communication system that uses fractal modulation to:

- overcome its error rates associated with low signal/noise ratios (SNRs), and
- optimise robustness of technique in presence of various noise fields.

For this application networks were also trained on $q(t)$, because of its following behaviour:

-independent of size of problem

-self similar at all scales

Figure 1.1 shows an overview of the research work carried out and the original contributions.

1.5.2 Outline

This thesis is organised as follows. Chapters 2 and 3 cover basics of digital signal processing techniques and fractal geometry. The bulk of information presented in these Chapters are mainly from lecture notes by Professor Blackledge on the MSc 'Digital Signal and Image Processing', which the author was required to attend all its modules as part of an induction course to the PhD.

Chapters 4 and 5 cover an introduction to various aspects of artificial neural networks.

Chapter 6 starts with an introduction to iterated function systems. Then using some general mathematical principles any link between fractals and artificial neural networks are explored.

In Chapter 7, employing a pattern recognition problem from a bio-chemical rig, important considerations in dealing with noise, incorporation of approximate models and ensuring optimum results are experimentally analysed. Joint publication based on this work, is included in Appendix C.

The practical work carried out in Chapter 8 and Chapter 9, investigate the applicability of artificial neural networks to a digital communication system which employs fractal modulation for secure transmission. Integrating neuro-filters with the digital communication system enables the system to operate consistently at lower signal/noise ratios (SNRs), making it commercially viable.

A summary of this research work and what it has revealed are provided in Chapter 10. The summary includes the conclusion of the work carried out and its main contributions. Further research directions are then discussed.

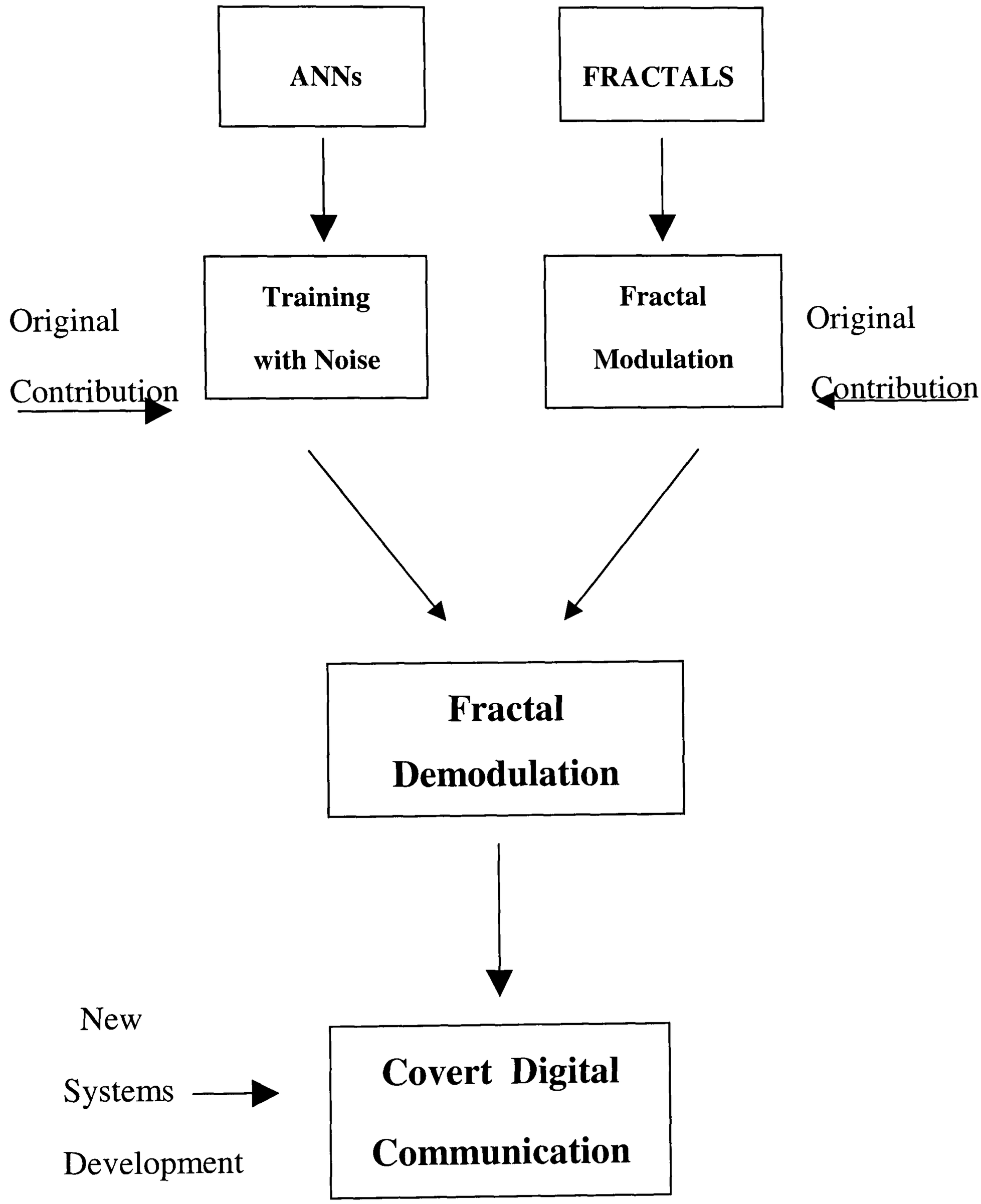


Figure 1.1
Overview of thesis with original contributions

2 SIGNAL PROCESSING TECHNIQUES

2.1 OVERVIEW

2.2 THE FOURIER TRANSFORM

The Fourier transform is just one of a variety of integral transforms but it has certain properties which make it particularly versatile and easy to work with. As a result, it is used extensively in many branches of science and engineering. Physically, the Fourier transform of a function provides a quantitative picture of the frequency content of the function. This is important in a wide range of physical problems and is fundamental to the processing and analysis of signals and images. For notation, the Fourier transform of a function f is usually denoted by the upper case F. If the function in question is already upper case F, then the transform is denoted by writing a tilde above this function. The transform is then given by $\tilde{F}$. To specify the forward or inverse Fourier transforms in full, the symbolic form $f(x) \Leftrightarrow F(k)$ is used which means that F is the Fourier transform of f and f is the inverse Fourier transform of F. Mathematical operations on f are referred to as operations in real, x-space or image space, and operations on F are referred to as operations in Fourier space or k-space.

- **Derivation of the Fourier transform pair from the complex Fourier series**

Let $c_n = F_n / 2l$,

then

$$f(x) = \frac{1}{2l} \sum_n F_n e^{\frac{inx\pi}{l}}$$

$$F_n = \int_{-l}^{l} f(x) e^{\frac{-inx\pi}{l}} dx$$

If $k_n = n\pi / l$ *and* $\Delta k_n = \pi / l$

then

$$f(x) = \frac{1}{2\pi} \sum_n F_n e^{ik_n x} \Delta k_n$$

$$F_n = \int_{-l}^{l} f(x) e^{ik_n x} dx$$

In the limit as $l \to \infty$,

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(k) e^{ikx} dx$$

$$F(k) = \hat{F}_1[f(x)] = \int_{-\infty}^{\infty} f(x) e^{-ikx} dx$$

Where $\hat{F}_1$ denotes 1D Fourier operator. *F(k)* is the Fourier transform of *f(x)* where *f(x) is* a non-periodic function. The variable *k* has dimensions that are reciprocal to those of the variable *x*. There are two important cases which arise in imaging science:

1. *x* - time in seconds; *k* - temporal frequency in cycles per second (Hertz). Here, *k* is referred to as the angular frequency which is given by 2π times frequency.

2. *x* – distance in meters; *k* – spatial frequency in cycles per meter. Here *k* is known as the wave number and is given by $k = 2\pi/\lambda$ where λ is the wavelength. *x* – distance in meters; *k* – spatial frequency in cycles per meter. Here *k* is known as the wave number and is given by $k = 2\pi/\lambda$ where λ is the wavelength.

The sufficient condition for the existence of the transform is that *f* is square integrable, i.e.,

$$\int_{-\infty}^{\infty} |f(x)|^2 dx < \infty$$

2.3 THE SPECTRUM

The Fourier transform of a function is called its 'spectrum'. It is a complex function that can be written in the form

$$F(k) = F_r(k) + iF_i(k)$$

Alternatively, using an Argand diagram representation

$$F(k) = A(k)e^{i\theta(k)}$$

where $A = |F| = \sqrt{F_r^2 + F_i^2}$

and $\theta = \tan^{-1}\left(\frac{F_i}{F_r}\right)$

The functions F_r and F_i are the real and imaginary parts of the Fourier transform respectively. If *f(x)* is a real valued function, then

$$F_r(k) = \int_{-\infty}^{\infty} f(x)\cos(kx)dx$$

and $F_i(k) = \int_{-\infty}^{\infty} f(x)\sin(kx)dx$

The functions F, A and θ are defined as follows:

F - complex spectrum

A - amplitude spectrum

θ - phase spectrum

In addition, the function $A^2 = |F|^2$ is known as the Power spectrum.

At $k = 0$, the value of the spectrum is called the DC level and is given by the integral of *f, i.e.*

$$\text{DC level} = F(0) = \int_{-\infty}^{\infty} f(x)dx$$

2.4 THE INVERSE TRANSFORM

The function $f(x)$ can be recovered from $F(K)$ by employing the inverse Fourier transform which is given by

$$f(x) = \hat{F}_1^{-1} F(k)$$

$$= \frac{1}{2\pi}\int_{-\infty}^{\infty} F(k)\exp(iks)dk$$

Where $\hat{F}_1^{-1}$ is the inverse Fourier operator and –1 is used to denote that this operator is an inverse operator and does not mean $1/\hat{F}_1^{-1}$.

- **Alternative definitions and representations**

It should be noted that some authorities define the forward and inverse Fourier transforms as

$$F(k) = \tfrac{1}{2\pi}\int_{-\infty}^{\infty} f(x)\exp(-ikx)dx$$

$$f(x) = \int_{-\infty}^{\infty} F(k)\exp(ikx)dk$$

or

$$F(k) = \tfrac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} f(x)\exp(-ikx)dx$$

$$f(x) = \tfrac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} F(k)\exp(ikx)dk$$

It is a matter of convention that F is called the Fourier transform of f when $-i$ occurs in the exponential and that f is the inverse Fourier transform of F when i appears in the exponential. The exact form of the forward and inverse Fourier transforms that are used does not matter. What does matter is that consistency with a given definition is maintained throughout a calculation. Here, the following Fourier transform pairs are used throughout

$$F(k) = \int_{-\infty}^{\infty} f(x)\exp(-ikx)dx$$

$$f(x) = \tfrac{1}{2\pi}\int_{-\infty}^{\infty} F(k)\exp(ikx)dk$$

2.5 THE DISCRETE FOURIER TRANSFORM

The forward and inverse Fourier transforms are not practical when dealing with values of $f(x)$ experimentally measured at discrete time intervals. In that case, the discrete Fourier transform (DFT) is required.

2.5.1 Derivation of the discrete Fourier transform from the complex Fourier series

The Complex Fourier Series over the range [-*l*, *l*] can be written as

$$f(x) = \frac{1}{2l}\sum_{n} F_n e^{\frac{inx\pi}{l}}$$

$$F_n = \int_{-l}^{l} f(x) e^{\frac{-inx\pi}{l}} dx$$

Consider the case where *f(x)* is uniformly sampled at points x_0, x_1, x_2, . . . x_{N-1} giving the discrete function or vector

$$f_m \equiv f(x_m); \quad m = 0, 1, 2, \ldots N-1$$

with sampling interval Δx.

Now, $x_m = m\Delta x$ and with $N = l/\Delta x$

$$f_m = \frac{1}{N}\sum_{n} F_n e^{\frac{i2\pi nm}{N}}$$

$$F_n = \sum_{m} f_m e^{\frac{-i2\ nm}{N}}$$

The crucial concept here is that the dot product of the data vector with sine and cosine waves at each of the frequencies under consideration is being computed. Also, with the forward and reverse transformations differing only in the sign in the exponent and in the division by the sample size for the inverse transform, only one subroutine is sufficient to do both operations.

2.5.2 Relationship between the DFT and the Fourier transform

- Fourier Transform Pair

$$F(k) = \int_{-\infty}^{\infty} f(x) e^{-ikx} dx \qquad \textit{and} \qquad f(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(k) e^{ikx} dk$$

- DFT pair

$$F_n = \sum_m f_m e^{\frac{-i2\ nm}{N}} \qquad and \qquad f_m = \frac{1}{N}\sum_n F_n e^{\frac{i2\pi mm}{N}}$$

Consider the following discretization of the Fourier Transform pair where Δx and Δk are sampling intervals

$$F(k_n) = \sum_m f(x_m) e^{-ik_n x_m} \Delta x \qquad and \qquad f(x_m) = \frac{1}{2}\sum_n F(k_n) e^{ik_n x_m} \Delta k$$

Writing $k_n = n\Delta k$ and $x_m = m\Delta x$ and comparing the results above with Discrete Fourier Transform pair, it can be seen that

$$\Delta k \Delta x = \frac{2\pi}{N}$$

$\therefore$

$$F_n = \sum f_m e^{-i2\pi nm/N}$$

$$f_m = \frac{1}{N}\sum_n F_n e^{i2\pi nm/N}$$

The interval Δk between the numbers F_m is related to the interval Δx between the numbers f_n by

$$\Delta k = \frac{2\pi}{N\Delta x}$$

2.6 THE FAST FOURIER TRANSFORM

The Fast Fourier Transform (FFT is an algorithm for computing the DFT with less multiplication and additions.

The standard form of DFT of an N-point vector is given by

$$F_m = \sum_n f_n \exp(-2 \quad inm/N)$$

where

$$\sum_n \equiv \sum_{n=0}^{N-1} \quad DC level (zero\ frequency\ component) occurs at F_0 - first value of F_m.$$

$$\sum_n \equiv \sum_{n=-N/2}^{(N/2)-1} for\ the\ optical\ form\ of\ DFT - DC\ level\ occurs\ at\ centre\ of\ array\ F_m.$$

Writing $W_N = exp(-2\pi i/N)$ *then* $F_n = \sum_n W_N^{nm} f_n$.

Computing the DFT in this form requires something on the order of N^2 operations. That is, multiplying an N–point vector f_n by a matrix of coefficients given by a complex constant W_N to the power of nm. This can be written in the form

$$\begin{pmatrix} F_0 \\ F_1 \\ \vdots \\ F_{N-1} \end{pmatrix} = \begin{pmatrix} W_N^{00} & W_N^{01} & .. & W_N^{0(N-1)} \\ W_N^{10} & W_N^{11} & .. & W_N^{1(N-1)} \\ \vdots & \vdots & . & \vdots \\ W_N^{(N-1)0} & W_N^{(N-1)1} & .. & W_N^{(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{pmatrix}$$

However, computing time could be halved by writing an N-point DFT in terms of two $N/2$-point DFTs. The Fast Fourier Transform Algorithm (FFT) is based on repeating this operation again and again until a single point DFT is obtained.

- Basic Idea

$$= \sum_{n=0}^{N-1} f_n e^{-2 \quad inm/N}$$

$$= \sum_{n=0}^{(N/2)-1} f_{2n} e^{-2 \quad i(2n)m/N} + \sum_{n=0}^{(N/2)-1} f_{2n+1} e^{-2 \quad i(2n+1)m/N}$$

$$= \sum_{n=0}^{(N/2)-1} f_{2n} e^{-2 \quad inm/(N/2)} + e^{-2 \quad im/N} \sum_{n=0}^{(N/2)-1} f_{2n+1} e^{-2 \quad inm/(N/2)}$$

$$= \sum_{n=0}^{(N/2)-1} f_{2n} W_{N/2}^{nm} + W_{N/2}^{m} \sum_{n=0}^{(N/2)-1} f_{2n+1} W_{N/2}^{nm}$$

- DFT of *N*-point array = DFT of even components + W_N^m × DFT of odd components.

  This can be written in the form $F_m = F_m^e + W_N^m F_m^o$ where e and o represent even and odd respectively

- The important thing to note here, is that the evaluation of F_m^e and F_m^o is over *N/2*-points - the *N/2* even components and the *N/2* odd components of the original *N*–point array.

- computing F_m^e and F_m^o requires only half the number of multiplication that are needed to compute F_m .

- Because the form of the expressions for F_m^e and F_m^o are identical to the form of the original N-point DFT, one repeat the idea and decompose F_m^e and F_m^o into even and odd parts producing a total four N/4-point DFT's and so on.

$$
\begin{array}{rcl}
 & F_m & \\
 & \Downarrow & \\
F_m^e & + & W_N^m F_m^o \\
F_m^{ee} + W_{N/2}^m F_m^{eo} & + & W_N^m \times (F_m^{oe} + W_{N/2}^m F_m^{oo})
\end{array}
$$

- Subdividing the data into odd and even components can continue until it gets down to the DFT of a single point.

- Because the data is subdivided into odd end even components of equal length an initial array of size $N = 2^k$ is required, where k is positive integer number.

- Computing the DFT in this way reduces the number of multiplications needed to $N \log_2 N$, a considerable improvement over N^2.

2.7 THE SHORT-TIME FOURIER TRANSFORM AND THE GABOR TRANSFORM

Mapping a time series vector to a time-frequency matrix, exposes the time-frequency structure of the series. Methods for performing such transforms include, the short time Fourier transform (STFT) and its special form, the Gabor transform.

2.7.1 The continuous STFT

The FT of a time domain signal, x(t), can be written as

$$F(k) = \int_{-\infty}^{\infty} x(t) e^{-ikt} dt$$

Consider the case where the frequency content of $x(t)$ is not constant. The FT can be modified to include time windowing $g(t)$.

$$F(\tau, k) = \int_{-\infty}^{\infty} x(t) e^{-ikt} g(t-\tau) dt$$

Under reasonable conditions, $F(\tau, k)$, the STFT within a continuous domain having a time dimension (τ) and a frequency dimension (k), captures all the information in $x(t)$.

2.7.2 The discrete STFT

The discrete short time Fourier transform (DSTFT) is simply the continuous STFT evaluated at discrete points, usually laid out on a equi-spaced lattice, in the time/frequency plane. The DSTFT of a signal $x(t)$ with a family of functions, $g_{pq}(t)$, where

p and q, represent the time and frequency dimension co-ordinates and

τ_0 and k_0 represent the spacing in the time and frequency dimensions respectively, is given by

$$F(p, q) = \int_{-\infty}^{\infty} x(t)\, g_{pq}(t)\, dt$$

where

$$g_{pq}(t) = g(t - p\tau_0)^{-ik_0tq}$$

producing equal lattice spacing

- **The inverse problem**

- In the continuous case, having $F(\tau, k)$, it is possible to fully capture $x(t)$ **[Daubechies**, 1990].

- In the discrete case, $F(p, q)$, information lost about $x(t)$ depends on how small are τ_0 and k_0. To accurately reconstruct discrete STFT, the concept of 'frames' is used.

  If $g_{pq}(t)$ constitute a frame, then all of the information in $x(t)$ is captured.

- **Necessary conditions to have a frame**

- $\tau_0 \cdot k_0 < 1$.

- As $t \rightarrow \infty$, the windows must rapidly decay.

- Windows must overlap, or the quality of the frame deteriorates.

Also there is the issue of aliasing. To avoid aliasing, for $x(t)$, sampled at time intervals t_0, the highest frequency handled is $1/(2\ t_0)$ cycles per time unit.

2.7.3 The Gabor transform

- The STFT and its special form the Gabor transform are influenced by the data window, $g(t)$.

- They make use of constant-shape resolution rectangles. Using the center and radius of the time and frequency domain windows, resolution rectangles can be visualised.

- The shape of the resolution rectangles can be controlled using the shape of $g(t)$.

- By letting $g(t)$ in the STFT be the Gaussian function, an optimal resolution product is produced. A STFT employing the Gaussian weight function is defined as the Gabor transform.

- **Influence by the position of the data window**

The position of the data window along the time axis defines a range of influence in the time domain.

- The centre of the area of influence, m_t, is

$$m_t = \frac{\int_{-\infty}^{\infty} t|g(t)|^2 dt}{\int_{-\infty}^{\infty} |g(t)|^2 dt}$$

- The distance from the centre over which the influence extends, the windows radius, Δ_t, is

$$\Delta_t = \sqrt{\frac{\int_{-\infty}^{\infty} (t-m_t)^2 |g(t)|^2 dt}{\int_{-\infty}^{\infty} |g(t)|^2 dt}}$$

- **Influence by the frequency content of $x(t)$**

The STFT is also influenced by the frequency content of $x(t)$ within the data window. The frequency domain window of a time domain window is the FT of the time domain window. With $G(f)$ defined as the Fourier transform of $g(t)$

$$m_f = \frac{\int_{-\infty}^{\infty} f|G(f)|^2 df}{\int_{-\infty}^{\infty} |G(f)|^2 df}$$

$$\Delta_f = \sqrt{\frac{\int_{-\infty}^{\infty} (f-m_f)^2 |G(f)|^2 df}{\int_{-\infty}^{\infty} |G(f)|^2 df}}$$

- **The Heisenberg uncertainty principle**

The Heisenberg uncertainty principle places a lower limit on the area of the resolution rectangle.

$$\Delta_t \Delta_f \geq 1 / 4\pi$$

Δ_t is the windows radius, i.e. the distance from the centre over which the influence extends, in the time domain and Δ_f is the windows radius in the frequency domain.

- **Optimal resolution**

An optimal resolution of the Gabor transform, up to the Heisenberg limit is achieved by letting $g(t)$ in the STFT be the normalised and scaled Gaussian function given by

$$g_\sigma(t) = \frac{1}{\pi^{1/4}\sqrt{\sigma}} e^{\left[-\frac{t^2}{2\sigma^2}\right]}$$

and it's FT is

$$G_\sigma(f) = \sqrt{2\sigma}\,\pi^{1/4}\, e^{-2\pi^2\sigma^2 f^2}$$

where the scale factor σ is

$$\sigma = \sqrt{\frac{k}{2\pi}}$$

2.8 WAVELET TRANSFORMS

2.8.1 Foundation

- Using the DSTRT, the time/frequency structure of a signal $x(t)$ is exposed by convolving $x(t)$ with $g(t)$

$$F(p,q) = \int_{-\infty}^{\infty} x(t)\, g_{pq}(t)\, dt$$

$g(t)$ producing equal lattice spacing is defined as

$$g_{pq}(t) = g(t - p\tau_0)^{-ik_0tq}$$

where

p and q determine discrete lattice points within a domain having time and frequency dimensions τ *and k respectively*

- Many practical problems require closer time-domain lattice-point spacing, i.e. finer time resolution, for higher frequencies.

- To locate higher frequency events more accurately in time, alternative tools, which unlike the STFT and the Gabor transform, can deal with the concept of 'scale', are required.

Rather than using frequency, k, as in the STFT and the Gabor transform, wavelets mostly use a scale dimension, ξ.

2.8.2 The continuous wavelet transform

The continuous wavelet transform (CWT) of a signal $x(t)$ within a domain having, a time dimension τ and a scale dimension ξ is given by

$$CWT(\tau,\xi) = \int_{-\infty}^{\infty} x(t)\, h^{\tau,\xi}(t)\, dt$$

Each family member $h^{\tau\xi}$ is given by shifting in time by τ and scaling by ξ

$$h^{\tau,\xi}(t) = \frac{1}{\sqrt{\xi}} h\left(\frac{t-\tau}{\xi}\right)$$

where

$\sqrt{\xi}$ *is to maintain the energy of the wavelet costant*

- Traditionally, wavelets are normalised according to

$$\int_{-\infty}^{\infty} |h(t)|^2 dt = 1$$

- Also it is vital for wavelets to have a mean of zero

$$\int_{-\infty}^{\infty} h(t)\, dt = 0$$

2.8.3 The discrete wavelet transform

For the discrete wavelet transform (DWT), in a similar manner to the STFT in which each family member $g_{pq}(t)$ was defined as

$$g_{pq}(t) = g(t - p\tau_0)e^{-ik_0tq}$$

each wavelet function corresponding to a point in the time/scale lattice, is given by

$$h_{pq}(t) = \frac{1}{\xi_0^{q/2}} h\left(\frac{t}{\xi_0^{q}} - p\tau_0 \right)$$

2.8.4 Quality of representation

- If $h_{pq}(t)$ constitute a frame, then all of the information in $x(t)$ is captured, but due to the time/scale lattice not being equally spaced, finding out if there is a frame and knowing its quality is relatively difficult. As a result, it is best to use mother wavelets whose frame quality associated with various values of τ_0 are known.

- Useful necessary conditions analogous to that in the DSTFT, e.g. $\tau_0 k_0 < 1$, do not exist. Despite this, the basic principles concerning a frame are similar to that in the DSTFT, and it is possible to construct a frame for any τ_0 and ξ_0.

- It is convenient, with regard to sampling, to choose $\xi_0 = 2$. But for some mother wavelets, $\xi_0 = 2$ is too large to generate a good frame, as τ_0 would have to be impracticably small. As a result more than one mother wavelet is required. The general form of voices is given by

  $$h^j(t) = 2^{-j/N} h(2^{-j/N}t), \qquad j = 0, \ldots, N\text{-}1$$

  This gives N lattices, lining up in time and shifted in frequency dimension.

2.8.5 The Morlet wavelet

- This is a family of variable width frame functions that is nothing but scaled versions of one 'mother' shape.

- This complex-valued wavelet can be considered as a modified Gabor function. The Gabor function was given as

$$g_\sigma(t) = \frac{1}{\pi^{1/4}\sqrt{\sigma}} e^{\left[-\frac{t^2}{2\sigma^2}\right]}$$

The Morlet wavelet is expressed by

$$h(t) = \pi^{-\frac{1}{4}}(e^{2\pi ikt} - e^{-k^2/2})e^{\left[-\frac{t^2}{2}\right]}$$

This is practically the same as the Gabor function with σ =1 and with a mean of zero. The term that centres the function is given by

$$(e^{2\pi ikt} - e^{-k^2/2})$$

- In practice, when applied to discrete sampled data, a value of $k > 0.7$ is used to avoid significant distortion of the shape of the mother wavelet. Additionally, at all scales, the normalisations of zero-centre and of unit energy are preserved. Therefore, as long as the sampling rate does not violate the Nyquist limit, there would be no trouble in regards to interactions between the oscillation of the wavelets and the spacing of samples.

2.8.6 The mexican hat wavelet

- This wavelet is not a complex valued wavelet like the Morlet wavelet. Rather, the Mexican hat wavelet as well as most other commonly used wavelets, are real-valued functions.

- It is one of the most used wavelets as it could provide a decent frame. It has a relatively wide window width in the frequency domain. Hence, for $\xi_0 = 2$, and the time domain spacing allowing overlaps, multiple voices are not necessary.

- It is the second derivative of the Gaussian function, with the sign reversed.

- The Mexican hat wavelet is given by

$$h(t) = \frac{2}{\sqrt{3}}\pi^{-\frac{1}{4}}(1-t^2)e^{-\frac{t^2}{2}}$$

and its FT is

$$H(f) = \frac{2}{\sqrt{3}}\pi^{-\frac{1}{4}}f^2e^{-\frac{f^2}{2}}$$

This Chapter presented basics of digital signal processing transfer techniques. The following Chapter presents fractal geometry.

3 FRACTAL GEOMETRY

3.1 INTRODUCTION TO FRACTALS

In the 17th century, Issac Newton explained the elliptical orbits of the planets around the Sun, discovered by Johannes Kepler, as following from the law of gravity. Associating the dynamics of a system with simplistic geometrical shapes implies that the system's future can be predicted from its past. The nature is full of systems that obey deterministic laws but behave unpredictably. Modelling deterministic chaos and describing natural objects such as the shapes of mountains and clouds require non-Euclidean structures, in particular fractal geometry. Fractal geometry plays two roles. It is the geometry of deterministic chaos and it can also describe the geometry of natural objects (Mandelbrot, 1977).

The term fractal was introduced in the 1970s by Mandelbrot. It is from the Latin 'fractus', which describes a broken up and irregular stone. Fractals are geometrical shapes that, unlike Euclidean shapes, are irregular all over, with the same degree of irregularity on all scales. A fractal object is (exactly, approximately, or statistically) self-similar. That is it looks the same when examined from far away or nearby. As one approaches it, the small pieces of the whole become well-defined objects whose shape is roughly that of the previously examined whole.

Examples of mathematical models of the way fractals work, includes the Sierpinski gasket and the Koch snowflake:

> The Sierpinski gasket, produces a simple fractal by breaking up a triangle into successively smaller ones.
>
> The Koch snowflake, or the Koch Island, was invented by Helge Von Koch in 1904 as an example of a 'nowhere smooth' curve. It is a fractal evolving from a simple triangle. It is produced by repeatedly replacing each side by 4 sides of length 1/3. As the number of repetitions approaches infinity, the curve will be infinitely long with an infinite number of discontinuities, with a notion of self-similarity at each level. However, the area enclosed is finite and very well defined (Blackledge J M, Turner M J, Andrews P R, 1998).

Consider the area enclosed by each level of the Island.

Defining the area of the original triangle as $A_0 = \Delta$

at level n

$$A_n = \Delta(1 + 1/3\sum_{i=0}^{n-1}(4/9)^i)$$

as $n \to \infty$

$$A_\infty = \Delta(1 + 1/3(1 - 4/9)) = 8/5\Delta$$

3.2 THE FRACTAL DIMENSION

Euclidean geometry uses the notions of *0, 1, 2* or *3* dimensions for a point, a line, a square or a cube respectively. For fractals, the counterparts of these dimensions are known as fractal dimensions, but, usually, their values are not whole numbers.

Self-similar objects are compounded by a parameter called the 'Fractal Dimension', or the' Similarity Dimension', D_s. Applied to a point, a line, a square or a cube, D_s simply gives 0, 1, 2, 3 respectively. In the case of a curve that is linearly self-similar fractal, ranging from being an almost smooth, one dimensional line to nearly plain filling and almost two dimensional, the corresponding value of D_s will range between $1 < D_s < 2$. The value of D_s characterises the fractal type (Table 3.1):

Fractal Type	Dimension
Dust	$0 < D_s < 1$
Signal	$1 < D_s < 2$
Image/Surface	$2 < D_s < 3$
Volume	$3 < D_s < 4$

Table 3.1

D_s is defined as

$$Nr^{Ds} = 1$$

or

$$D_s = -\frac{\ln N}{\ln r} = -\frac{\ln(Number\ of\ self-similar\ pieces)}{\ln(Magnification\ factor)}$$

N is the number of distinct copies of an object which has been scaled down by a ratio r in all co-ordinates.

There are two distinct types of fractals which exhibit this property:

1. Deterministic Fractals;
2. Random Fractals.

Deterministic fractals are objects which look identical at all scales. Each magnification reveals an ever finer structure which is an exact replication of the whole, i.e. they are exactly self similar. Random fractals do not, in general, posses such deterministic self-similarity; such fractal sets are composed of N distinct subsets, each of which is scaled down by a ratio r from the original and is identical in all statistical respects to the scaled original – they are statistically self-similar. Certain fractal sets are composed of the union of N distinct subsets, each of which is scaled down by ratio $r_i < 1,\ 1 \le i \le N$ from the original in all co-ordinates. The similarity dimension is given by a generalisation of $Nr^{Ds} = 1$, namely

$$\sum_{i=1}^{N} r_i^{Ds} = 1$$

A further generalisation leads to self-affain fractal sets which are scaled by different ratios in the different co-ordinates. The equation

$$f(\lambda x) = \lambda^H f(x) \quad \forall \lambda > 0$$

where λ is a scaling factor and H is the scaling exponent implies that a scaling of the x-coordinate by λ gives a scaling of the f co-ordinate by a factor λ^H. A special case occurs when $H = 1$; in which case, a scaling of x by λ produces a scaling of f by λ, i.e. $f(x)$ is self-similar.

Naturally occurring fractals differ from strictly mathematically defined fractals in that they do not display statistical or exact self-similarity over all scales but exhibit fractal properties over a limited range of scales.

3.3 BROWNIAN MOTION

- There are many examples in the field of physics, chemistry and biology of random processes.

- Brownian motion is a relevant mathematical model for many such physical processes.

- These processes display properties which are best described as fractal processes.

In Brownian motion, the position of a particle at one time is not independent of the particles motion at a previous time. It is the increments of the position that are independent. Brownian motion in 1 D is seen as a particle moving backwards and forwards on the x-axis for example. If the particle's position is recorded on the x-axis at equally spaced time intervals, then we end up with a set of points on a line. Such a point-set is self-similar. On the other hand, if the time is included as an extra co-ordinate and plot the particles position against time – (called the record of the motion) - a point- set is obtained that is 'self-affine'.

3.3.1 Diffusion as an example of Brownian motion

In this Section, an example of a physical process that has been modelled by Brownian motion is given. For a particle moving in 1 D (along the x-axis), consider the following model for its motion. At time interval τ a displacement (or increment) ξ is chosen at random from a Gaussian probability distribution given by

$$P(\xi,\tau)=\frac{1}{\sqrt{4\pi\rho\tau}}\exp\left(-\frac{\xi^2}{4\rho\tau}\right)$$

where ρ is the diffusion coefficient. The probability of finding ξ in the range ξ to ξ to $\xi+d\xi$ *is* $P(\xi,\tau)$ and the sequence of the increments $\{\xi_i\}$ is a set of independent Gaussian random variables. The variance of the process is

$$\left\langle \xi^2 \right\rangle = \int_{-\infty}^{\infty} \xi^2 P(\xi,\tau)d\xi = 2\rho\tau$$

where

$\langle\cdot\rangle$ *denotes the expectation.*

The position of the particle at time t is then

$$x(t) = \sum_{i=1}^{n} \xi_i$$

Normally, for convenience, x(0) = 0 is imposed.

3.3.2 Scaling properties

Suppose that we observe the motion not at intervals τ, but at intervals $\lambda\tau$ where λ is some arbitrary number. For example, if $\lambda = 2$, the increment ξ during time interval t to t+2τ will be given by $\xi=\xi_1+\xi_2$ where ξ_1 is the increment in time interval t to t+τ and ξ_2 is the increment in time interval t+τ to t+2τ. ξ_1 and ξ_2 are independent increments and hence the joint probability $P(\xi_1 : \xi_2, \tau)$, that the first increment is <u>in</u> the range ξ_1 to $\xi_1+d\xi_2$ and the second increment is in the range ξ_2 to $\xi_2+d\xi_2$ is given by

$$P(\xi_1 : \xi_2, \tau) = P(\xi_1, \tau)\, P(\xi_2, \tau)$$

Hence the probability density for ξ is given by integrating over all possible combinations of increments ξ_1 and ξ_2 such that $\xi = \xi_1 + \xi_2$, i.e.

$$P(\xi, 2\tau) = \int_{-\infty}^{\infty} P(\xi - \xi_1, \tau) P(\xi_1, \tau)\, d\xi_1$$

$$= \frac{1}{\sqrt{4\pi\rho 2\tau}} \exp\left(-\frac{\xi^2}{4\rho 2\tau}\right)$$

Therefore, if the particle is viewed with half the time resolution, the increments are still a random Gaussian process with $\langle\xi\rangle = 0$, but with variance now given by

$$\left\langle \xi^2 \right\rangle = 2 \times 2\rho\tau$$

i.e. twice the value obtained when the process is viewed at intervals τ

In general, for observations at time interval λ

$$P(\xi, \lambda\tau) = \frac{1}{\sqrt{4\pi\rho\lambda\tau}} \exp\left(-\frac{\xi^2}{4\rho\lambda\tau}\right)$$

where $\langle\xi\rangle = 0$ *and* $\langle\xi^2\rangle = \lambda \times 2\rho\tau$.

Note, that with $\tau=\lambda\tau$ *and*

$$\xi = \lambda^{\frac{1}{2}}\xi$$

$$P(\xi = \lambda^{\frac{1}{2}}\xi, \tau = \lambda\tau) = \lambda^{\frac{1}{2}}P(\xi, \tau)$$

which is the scaling relation for the probability density.

The above equation shows that the Brownian process is invariant in its statistical distribution under a transformation that changes the time scaled by a factor λ and the length scale by a factor $\lambda^{1/2}$. The name given to such transformations is *affine* and the curves or records that reproduce themselves in some sense under transformations of this type are called *self-affine*.

Finding the probability distribution for the particle position x(t),

note that:

$$P[x(t) - x(t_0)] = P[x(t) - x(t_0), t - t_0]$$

gives

$$P[x(t) - x(t_0)] = \frac{1}{\sqrt{4\pi\rho|t - t_0|}} \exp\left(-\frac{[x(t) - x(t_0)]^2}{4\rho|t - t_0|}\right)$$

and satisfies the scaling relation

$$P[\lambda^{-\frac{1}{2}}, x(\lambda t) - x(\lambda t_0)] = \lambda^{-\frac{1}{2}} P[x(t) - x(t_0)]$$

$x(t_0)$ is the particle's position at some arbitrary reference time.

Finally expressions for the mean, mean absolute and the variance of the particle's position can be derived and are given respectively by

$$\langle x(t) - x(t_0) \rangle = 0$$
$$\langle |x(t) - x(t_0)| \rangle = \sqrt{\frac{4\rho}{\pi}} |t - t_0|^{\frac{1}{2}}$$
$$\langle [x(t) - x(t_0)]^2 \rangle = 2\rho|t - t_0|$$

.

For ξ a normalised independent Gaussian random process

$$x(t) - x(t_0) \propto \xi |t - t_0|^{\frac{1}{2}}$$

This result can be generalised to the form

$$x(t) - x(t_0) \propto \xi |t - t_0|^{H}, \; 0 < H < 1$$

which provides the basis for Fractional Brownian Motion. Fractional Brownian Motion is an example of statistical fractal geometry and is the basis for the experiments Chapters 8 and 9.

3.3.3 White noise and fractal noise

- The relationship between white noise and fractal noise can be considered in terms of fractional differentiation and fractional integration

Fractional integration	*of white noise*	*gives fractal noise*
Fractional differentiation	*of fractal noise*	*gives white noise*

- Note that the normal interpretation of integration being a 'smoothing' process, and differentiation be a 'roughening' process, for integer order, continues to hold for fractional order.

- Fractal noise and white noise, as well as fractional differential equations, and random scaling fractal signals are discussed in Section 8.4 (Modeling and computing noise).

The following two Chapters briefly cover various aspects of ANNs. Links between Fractals and ANNs are then covered in Chapter 6.

4 MODELS AND ALGORITHMS

4.1 THE BIOLOGICAL MODEL

The Processing Element for most neural models is based on a simplified view of a neural cell; a slow but complex analogue device with many functions including some which take place outside the cell. The signals produced by neural cells or neurones are virtually indistinguishable, regardless of the species producing them. There are of the order of 10^{11} neurones in the human nerve system.

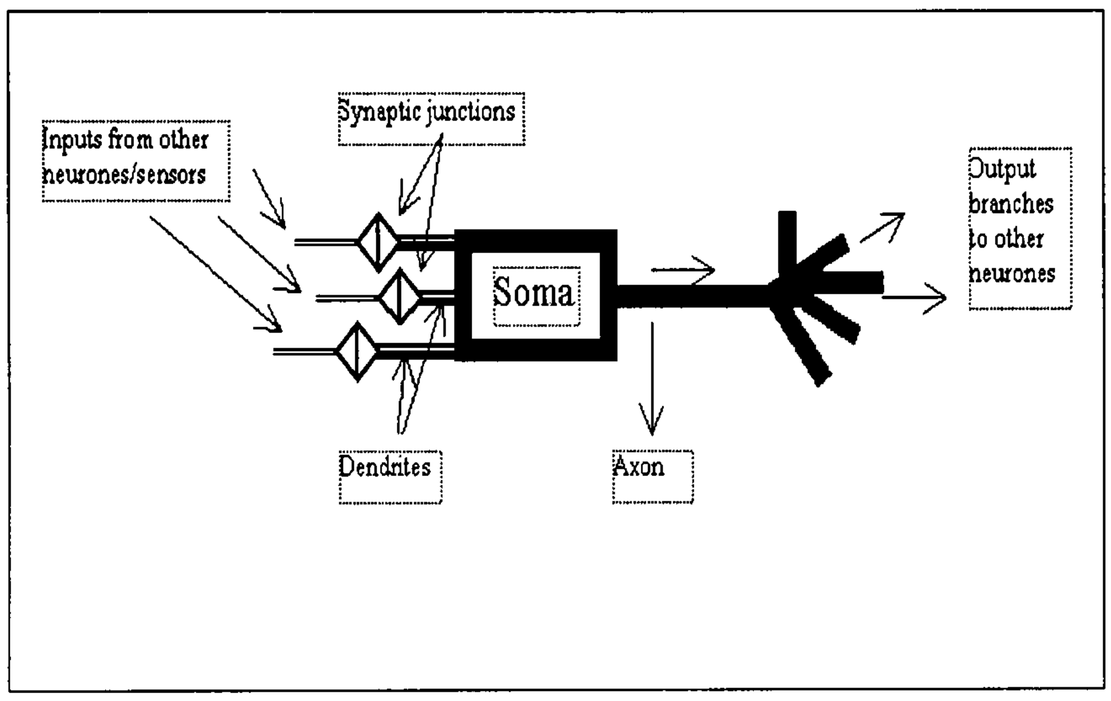


Figure 4.1
A simplified biological neurone

A typical neurone consists of the following sections, see Figure 4.1 for a simplified scheme:

- Soma is the cell body. It provides aggregation, thresholding, and non-linear activation to inputs. Soma has two types of extensions: dendrites and the axon. Dendrites are the input lines of the neurone. Each soma receives, on the average 10,000 signals from other neurones. The overall role of the soma is to perform a spatio-temporal weighted aggregation, often a summation, of all these inputs. If this weighted aggregation exceeds a certain threshold then the neurone will fire. The output signal undergoes a non-linear transformation prior to transmission along the axon to the other neurones. The axon, usually divides up into branches along which a common output is sent to other neurones or muscle cells. Each neurone requires 2 ms in order to fire, i.e., a bandwidth of the order of 500 Hz, and the firing or response of a neurone is largely a random variable. The only way to achieve fast and efficient information processing using such slow and unpredictable processors, is to employ large numbers of them.

- The synapse is the contact terminal between the end of an axon branch and another neurone. It converts electrical charges into chemicals that pass messages to the neurones. A neurone can receive several thousand inputs through the synapses from other neurones, and transmit modified, weighted, versions of these signals to the soma via the dendrites. Each synapse is a storage element that contains some attribute of the past experience. Synapses learn by continuously adapting their strength, or weights, to the new inputs.

In summary, the processing of information within the biological neurone involves two distinct operations:

1. a somatic operation, providing aggregation, thresholding, and non-linear activation, to produce an output signal if the weighted aggregation of the inputs exceeds a certain threshold, and

2. a synaptic operation, assigning a relative weight, or significance, to each incoming signal according to the past experience stored in the synapse.

The study and modelling of the biological functions come under the general heading of neural science; for more details on neurones see texts on neuro-physiology (Diamond, 1985).

4.2 THE ARTIFICIAL MODEL

In comparison to the biological model an artificial neurone or processing element (PE) is a simple device. In a similar manner to biological neurones they use many different methods of input signal combining techniques. The processing element maps an input pattern of signals to an output pattern. In a typical artificial model, a weighted sum of the inputs constitutes the argument of a non-linear 'activation function'. Figure 4.2 shows the functional model of the processing element, which consists of the weighted input connections, the summation function, and a non-linear activation function with w_0 threshold which generates the processing element's output. Linear activation functions are only used in the output processing elements of a network, since a sequence of linear functions can be reduced to a single linear function.

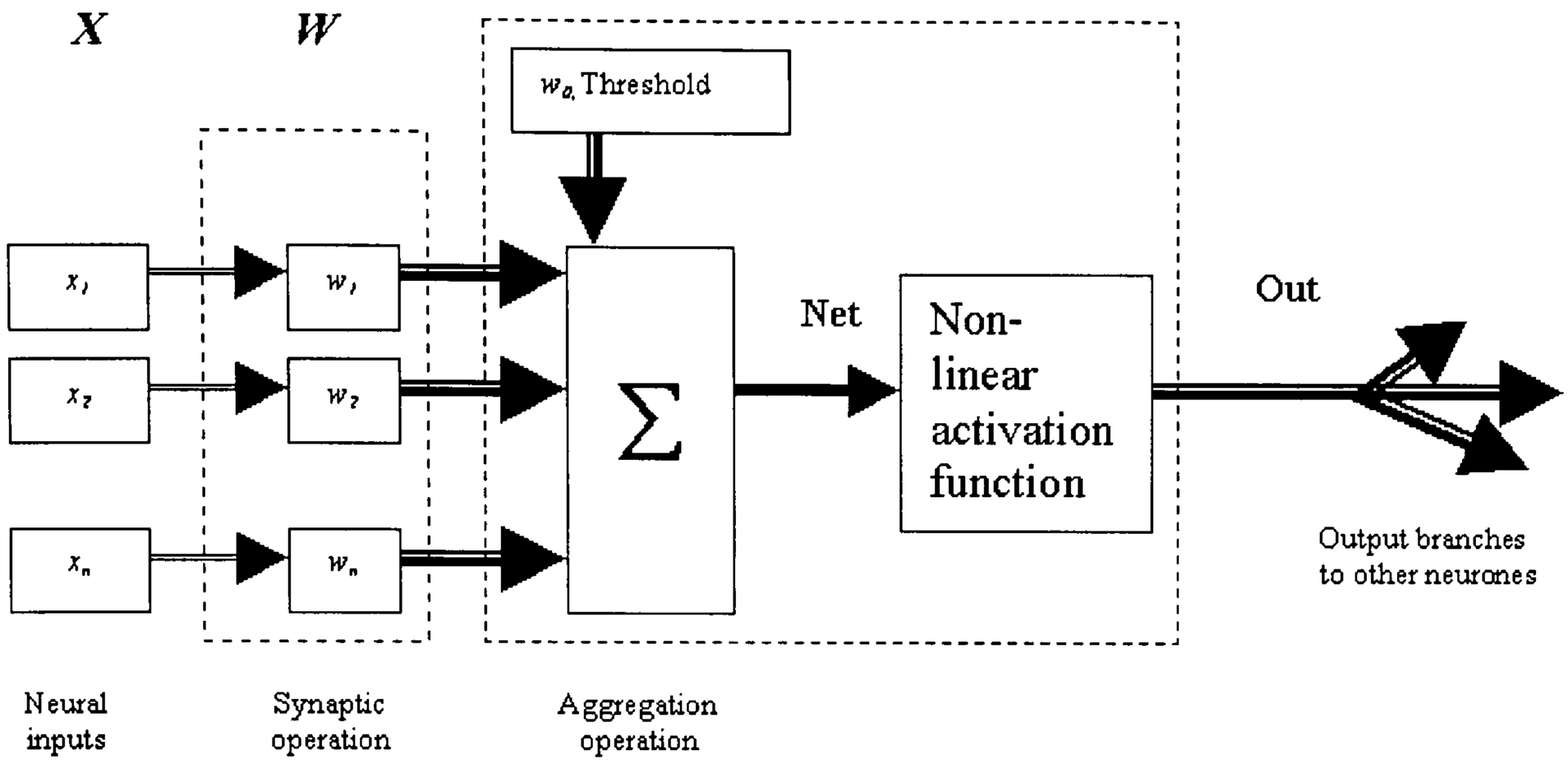


Figure 4.2
The functional model of the processing element

The function of the processing element for most models is generally expressed in the form:

$$\mathbf{Out} = F\left[\sum_{i=1}^{n} w_i x_i + w_0\right]$$

$$\mathbf{Out} = F[\mathbf{Net}]$$

where

Net = ***W.X*** *in vector notation*

Out = *is the neuro- output,*

$\boldsymbol{X} = [x_1, \ldots, x_n]$ = *represents the n inputs,*

$\boldsymbol{W} = [w_1, \ldots, w_n]$ = *are the synaptic weights,*

$F[\cdot]$ = *some non-linear activation function with* w_0 *threshold.* w_0 *may be introduced by employing an additional input* x_0 *equal to* +1, *so*

$$\mathbf{Out} = F\left[\sum_{i=0}^{n} w_i x_i\right]$$

4.3 MATHEMATICAL REPRESENTATION

From a signal-processing viewpoint, to perform computational tasks, such as pattern recognition, learning, and storage of past experience, the two distinct operations of the biological neurone can be interpreted as the following key operations:

1. a confluence operation between new inputs and stored knowledge (past experience), and
2. a non-linear bounded mapping to the aggregated signal, as shown in Figure 4.3.

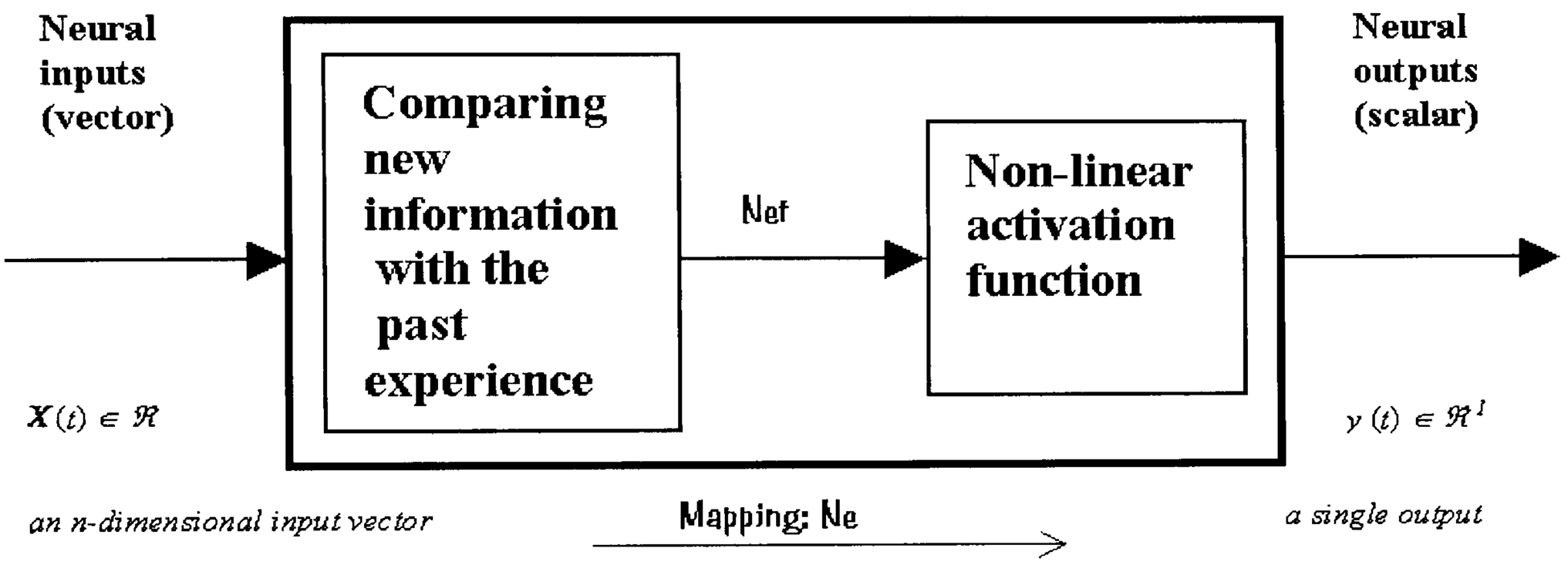


Figure 4.3

A (PE) receives *an n-dimensional input vector*

$$\boldsymbol{X}(t) = [x_1(t), x_2(t), x_3(t), \ldots\ldots, x_i(t), \ldots\ldots, x_n(t)]^T \in \Re^n$$

and ⟶ *yields a scalar output*

-(*transmission to many other PEs*)

The information processing ability of a processing element (PE) *can be represented by the non-linear mapping function* N_e

$$X(t) \in \Re^n \xrightarrow{\text{non-linear mapping function Ne}} y(t) \in \Re^1$$

i.e. The output at time t

is

$$y(t) = N_e[X(t) \in \Re^n] \in \Re^1$$

X (t) and y(t) should be viewed in terms of digital graphs or vectors in n and 1D *space*

- Representation of multi input/output systems taking the above form, with, and $X(t)$ representing the input, and $y(t)$ representing the output at time t is currently well established.

Mathematically, the neuronal non-linear mapping function can be divided into two parts: (1) confluence, ©, and (2) non-linear activation.

In order to account for thresholding in the confluence operation, the vectors of inputs and weights can be defined as follows:

$$X_a(t) = [x_0(t), x_1(t), x_2(t), ..., x_i(t), ..., x_n(t)]^T \in \Re^{n+1}$$
$$x_0(t) = 1$$

and

$$W_a(t) = [w_0(t), w_1(t), w_2(t), ..., w_i(t), ..., w_n(t)]^T \in \Re^{n+1}$$

where

$w_0(t)$ *introduces a thresholding, bias, term in the confluence operation which is a similarity measure between* $X_a(t)$ *and* $W_a(t)$

The two basic mathematical operations of a PE, i.e. the confluence operation which provides a measure of similarity, and the non-linear activation operation which performs a non-linear mapping on the similarity measure, are further described in the following sections.

4.4 MEASURE OF SIMILARITY

- Biologically, the confluence operation, ©, or measure of similarity, provides the weighting, aggregating, and thresholding operations to the neural inputs. It represents the weighting of $X_a(t) \in \Re^{n+1}$ with the accumulated knowledge stored at the synapses, $W_a(t)$, and the spatio-temporal aggregation of these weighted inputs, as performed by the soma. The synaptic weighting assigns a relative weight to each incoming signal component $x_i(t)$

according to an attribute of the past experience (knowledge or memory) stored in synaptic weight $w_i(t)$.

- Mathematically, this operation can be viewed as a linear weighted mapping

  from the $(n+1)$ *- dimensional neural input space*

  $$X_a(t) \in \Re^{n+1}$$

  to the one-dimensional space

  $$u(t) \in \Re^1.$$

- The synaptic (weighting) and somatic (aggregation and thresholding) linear mapping can be modelled as

  $$u(t) = W_a(t) \copyright X_a(t)$$

  This represents a measure of the similarity between input vector, $X_a(t)$, and synaptic weight vector, $W_a(t)$.

- The two types of similarity measures, used by most processing elements, are:

  (1) the scalar (inner) product of the vectors $X_a(t)$ and $W_a(t)$, and

  (2) the Euclidean distance between vectors $X_a(t)$ and $W_a(t)$.

4.4.1 Inner product

The inner product of $X_a(t)$ and $W_a(t)$ is the most common measure of similarity used. Geometrically, this is defined as the projection of the input vector $X_a(t)$ (new information) onto the weight vector $W_a(t)$ (the accumulated knowledge), as illustrated in Figure 4.4; i.e.,

$$u(t) = W_a(t)^T X_a(t)$$

$$= \sum_{i=0}^{n} w_i x_i$$

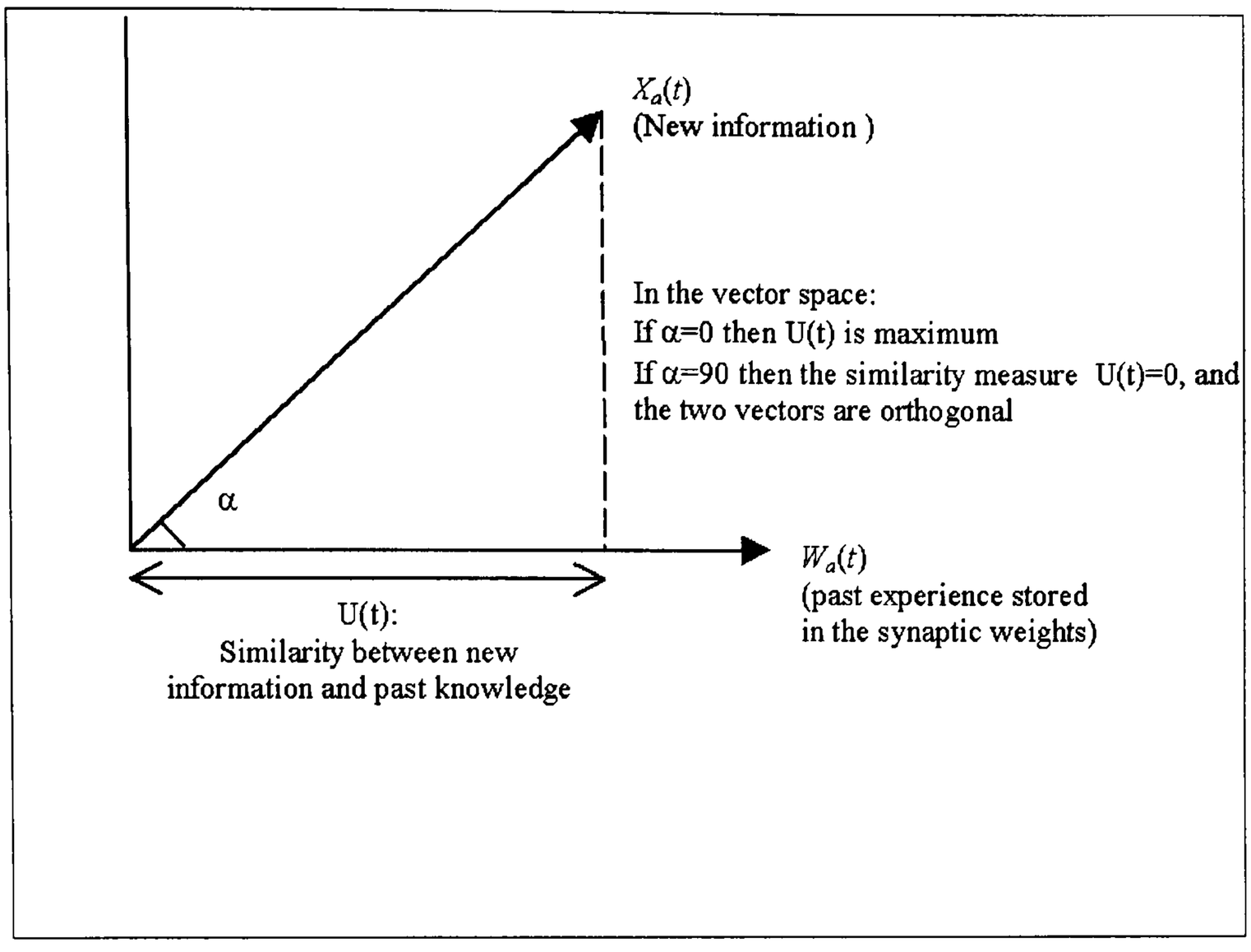


Figure 4.4
A measure of similarity based on projection (inner-product)

4.4.2 The Euclidean distance

- The processing elements of some networks, e.g. the radial basis function, employ the distance measure for describing the confluence between the inputs $X_a(t)$ and weights $W_a(t)$, as shown in Figure 4.5. The Euclidean distance between the new neural information $X_a(t)$ and the accumulated knowledge $W_a(t)$ is given by

$$D = \beta\sqrt{\left[W_a(t) - X_a(t)\right]^T \left[W_a(t) - X_a(t)\right]} \in \Re^1$$

where β is a normalisation constant such that $0 \leq D \leq 1$

The measure of similarity may then be defined as

$$u(t) = \left[1 - D\right]$$

i.e. *if $D = 0$, then $u(t) = 1$ and*

if $D = 1$, then $u(t) = 0$

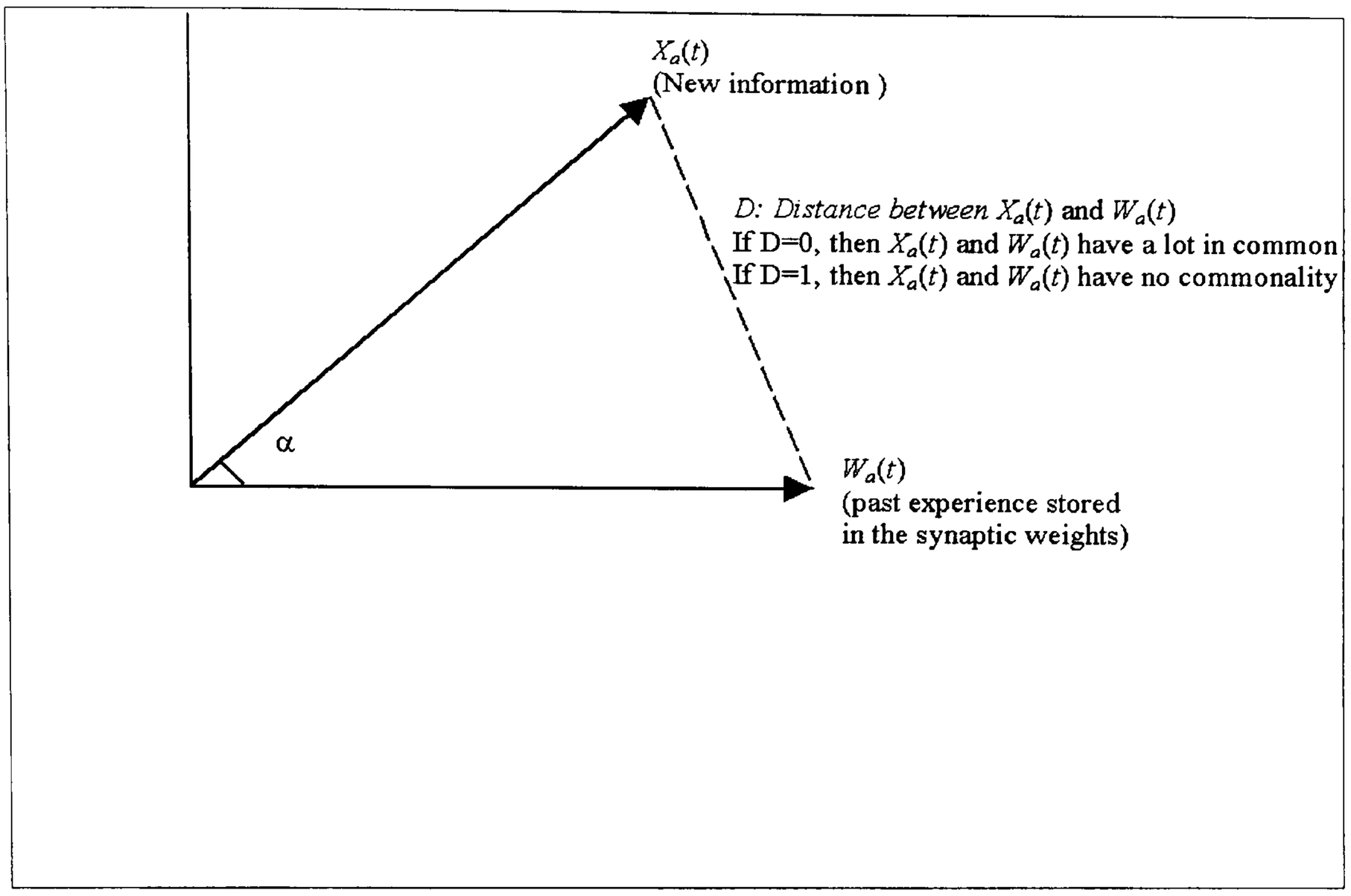


Figure 4.5
Euclidean distance measure of similarity between the new information and the previously accumulated synaptic

4.5 THE NON-LINEAR ACTIVATION FUNCTION

The somatic non-linear activation function $F[\cdot]$ maps the confluence value $u(t)$ to a bounded output. In general, the output is in the range of (0, 1) for unipolar signals and (-1, 1) for bipolar signals. The activation operator transforms the aggregated $u(t)$ into a bounded output $y(t)$; that is,

$$y(t) = F[u(t)]$$

$$y(t) = F[W_a(t) \circledcirc X_a(t)]] \in \Re^1$$

Many different forms of mathematical functions can be used to model the activation function. Early artificial neurones used a simple activation function with an output of one for firing and zero for not reaching a predefined threshold. This was then improved by making the function continuous to give an output equal to the net input minus the thresold. If the threshold was not reached, the processing element would not fire. Figure 4.6 shows some of the possible geometrical shapes: (1) Linear; (2) Piecewise linear; (3) Threshold (or hard limiter); (4)

Unipolar Sigmoidal; (5) **B**ipolar Sigmoidal; (6) Unipolar multimode Sigmoidal; (7) Radial basis function (RBF).

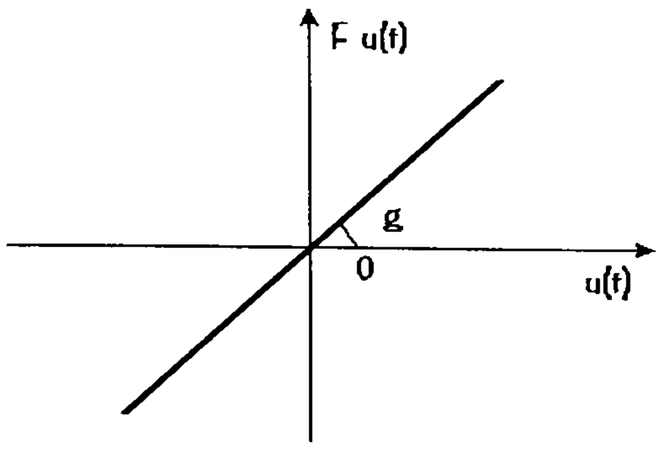
The sigmoid function, which squashes the output to values between 0 and 1 and the hyperbolic tangent, which squashes the output between -1 or 1, are frequently called the logistic activation function. The logistic activation function is a widely used activation function, and is generally regarded as the most effective. Using the logistic function, the same network may handle both small input signals (requiring high gain) and large input signals (requiring smaller gain) and enables processing elements to perform over a wide range of input levels. A main reason for the widespread use of this continuously differentiable function is that it has the advantage of having a simple derivative. Algorithms, such as back-propagation require a simple derivative of the function, and with an S-shed function mathematically approximated for example as

$$f(x) = \frac{1}{(1+e^{-x})}$$

the simple derivative fulfils this requirement, i.e.,

$$f'(x) = f(x)(1-f(x))$$

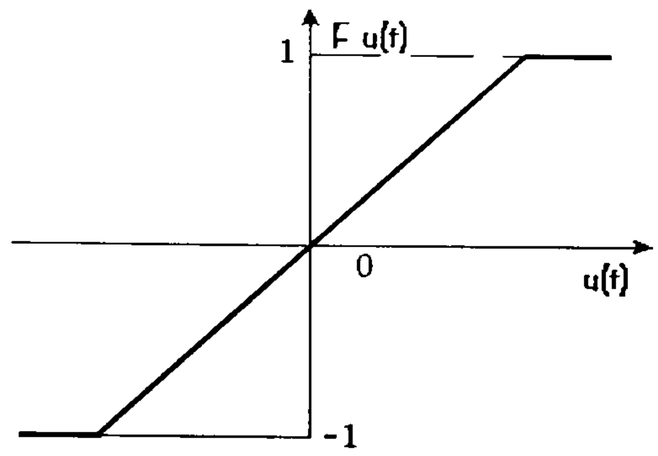
It may also be noted that the logistic function is used in many electronic circuits (called the Schmitt Trigger). The Schmitt Trigger's linear behaviour over most of its operating range is an important characteristic of this device.



(1) *Linear*

$F[u(t)] = g\,u$

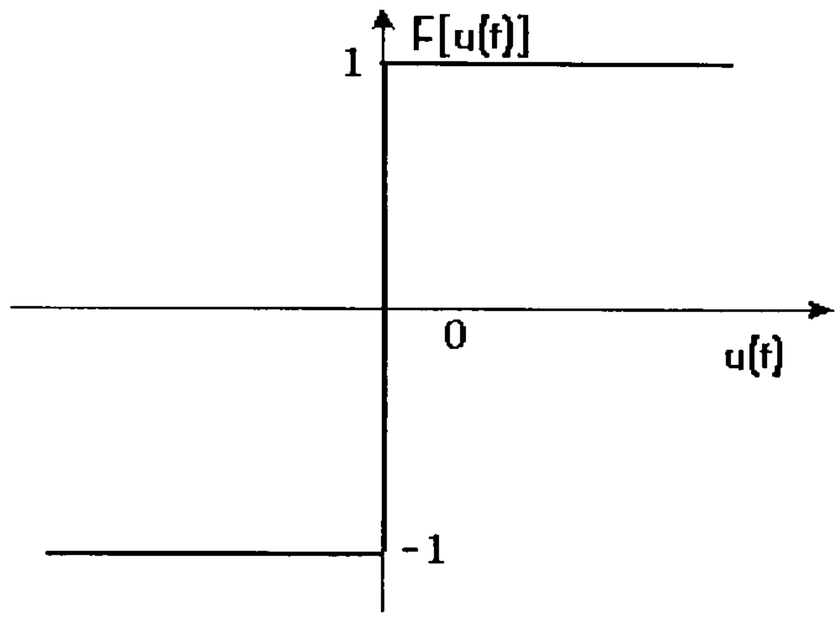
$g > 0$, *activation gain*



(2) *Piecewise linear*

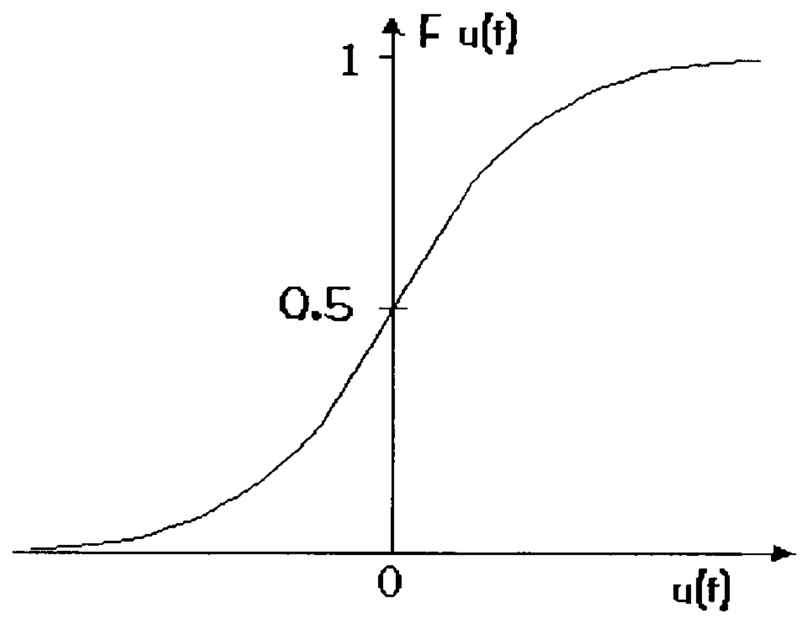
$$F[u(t)] = \begin{cases} +1 & if\ gu > 1 \\ gu & if\ |gu| < 1 \\ -1 & if\ gu > 1 \end{cases}$$

$g > 0$, *activation gain*



(3) *Hard* lim *iter*

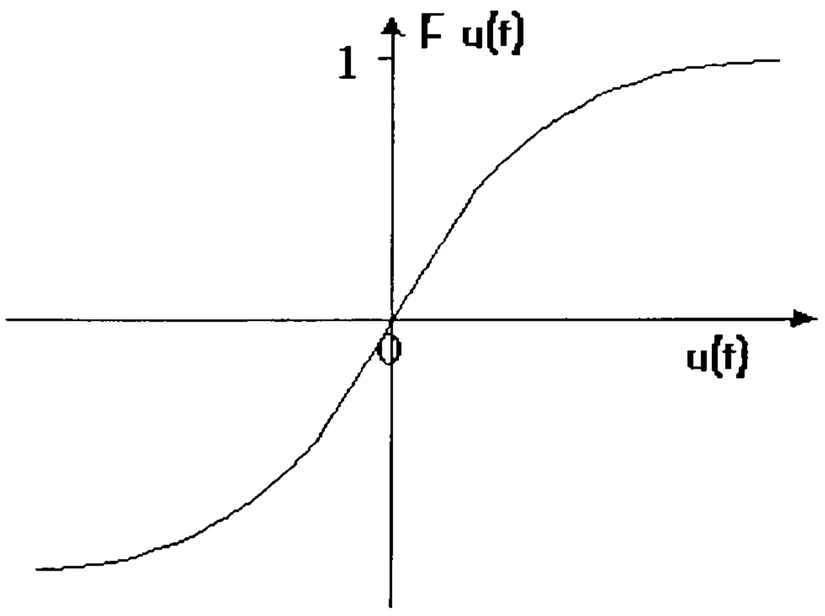
$F[u(t)] = \mathrm{sgn}[u]$



(4) *Unipolar sigmoidal*

$F[u(t)] = \frac{1}{1+e^{(-gu)}}$

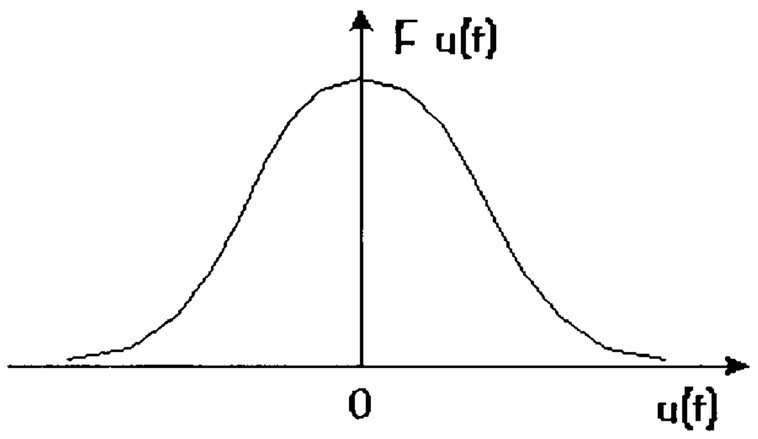
$g > 0$, *activation gain*



(5) *Bipolar sigmoidal*

$F[u(t)] = \tanh[gu(t)]$

$g > 0$, *activation gain*



(6) *Radial basis function* (*RBF*)

$F[u(t)] = \exp(u(t))$

$$u(t) = \frac{1}{2}\left[\frac{-\sum_{i=0}^{N}(w_i(t)-x_i(t))^2}{2\sigma^2}\right.$$

Figure 4.6
Examples of typical activation operators F [·]

4.6 NETWORK TOPOLOGY

It is generally assumed that the complexity of an artificial neural system stems from the way in which the processing elements can interact and the number and type of processing elements used. The processing elements are usually arranged in groups or layers. In single layer networks, a group of processing elements is connected to itself, i.e. the output is connected to the input. Such systems are referred to as auto associative systems. Generally if there is feedback from the outputs of the network to the inputs, the network is said to be a "recurrent network" (or non-recurrent if there is no feedback). Although there are many disadvantages with single layer networks, they have been used extensively in research and development as the basic core of modern neural networks.

Multi-layer networks consist of a number of single layers connected to each other, fully or selectively. In its simplest form, the output of the first layer is the input to the second layer and so on. Sometimes all of the layers between the input and output are referred to as hidden layers.

Since the 1960's many algorithms have been developed to train such networks. The flow of information is usually from the input to the output and is referred to as feed forward. A multiple feed forward network may receive weighted inputs from all the input units of the previous layer and send its output to all units of the next layer.

Sometimes, the training and learning procedures are improved by feeding back the results from a succeeding layer to a previous layer. The network in this case is referred to as a feedback network. There is also a concept referred to as "Locally Interconnected Network", such as that used in the Mead Silicon Retina System, in which the processing elements connect only to their nearest neighbours.

Among the algorithms developed for multi layer systems, back propagation has had a significant effect on the field. Although the training process is slow and uncertain, as a general purpose systematic method, it has allowed the development of a range of neural network applications.

Statistical methods (e.g. Boltzmann Training) have also been used extensively for training networks. They are generally thought of as being straight forward methods to use in practice.

4.7 ALGORITHMS

The weighting and spatio-temporal aggregation operations performed by the synapses and soma, respectively, provide a similarity measure between the input vector $X_a(t)$ (new neural information) and the synaptic weight vector $W_a(t)$ (accumulated knowledge base). When a new input pattern that is significantly different from the previously learned patterns, is presented to the network, the similarity between this input and the existing knowledge base is small. As the neural network learns this new pattern (by changing the strength of the synaptic weights) the distance between the new information and accumulated knowledge decreases

Most neuro-based soft computing structures undergo a learning procedure during which the synaptic weights are adapted. Algorithms for varying these connection strengths such that learning ensues are called learning rules, each cycle of presentation of all cases is usually referred to as a learning epoch. The objective of learning rules depends on the application. For example, the objective in pattern classification from sample data is to classify and predict successfully on new data.

In the application of soft computing techniques, with artificial neural networks as a base, the main algorithms may be broadly categorised as error based or output based.

4.7.1 Error based algorithms

Error-based learning algorithms need desired responses or data labelled with target results. If desired (target) results are unknown, then error based learning algorithms are useless. In an error based scheme, schematically shown in Figure 4.7, an external reference is compared with the obtained response. The error is then used for modification, in order to improve the system performance. Examples of error based algorithms include error correction (e.g. least mean square 'LMS', back-propagation), and Stochastic (e.g. simulated annealing).

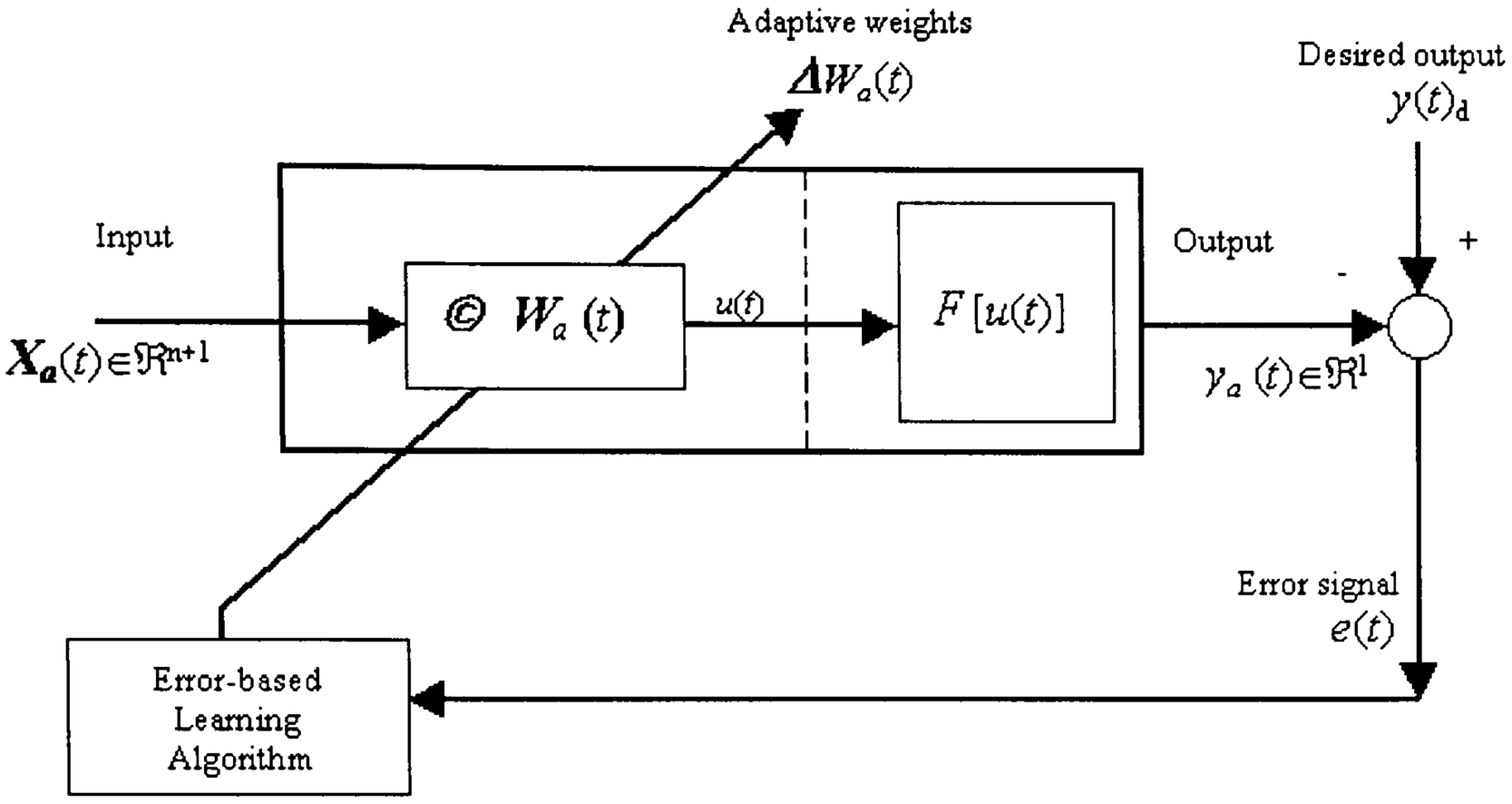


Figure 4.7
An error based (supervised) learning scheme

A general equation for the error-based learning algorithm is

$$w_i(t+1) = w_i(t) + \Delta w_i(t)$$

where

$$\Delta w_i(t) = \mu x_i(t)\left[y_d(t) - y(t)\right]$$

and

w_i *is the synaptic weight corresponding to*

$x_i(t)$, *the input. The parameter*

$\Delta w_i(t)$ *is the change in*

$w_i(t)$ *synaptic connection over an instant in time,*

μ *is the learning rate,*

$y_d(t)$ *is the desired neural output, and*

$y(t)$ *is the actual neural response.*

The proper selection of μ is of critical importance in these learning rules. A very small value of μ will result in extremely slow learning. On the other hand, a large value of μ will make learning faster, but it may also result in oscillations or make the system unstable.

Stochastic error based learning algorithms make random changes to the weights. A resultant 'energy' created by this change is then determined. The weight changes are retained if the 'energy' is lowered. A pre-defined probability distribution may be used to keep the weight change, even if energy is not lowered.

4.7.2 Output based algorithms

If desired, or target, results are unknown, then error based learning algorithms cannot be used and output based learning algorithms become useful. Such algorithms don't employ any external reference signal. Self organisation principles and internal control mechanisms are generally used to classify the input data and to discover collective properties. The two important forms of output based, or unsupervised, learning algorithms are Hebbian learning and competitive learning.

Hebbian learning, Figure 4.8, is guided by the neural output rather than output error as in a supervised learning scheme. A large number of variations of the Hebbian learning rule are described in the literature (Caudill and Butler, 1992). They are all guided by the neural output. A simple Hebbian learning rule used to describe the correlation of the input $x_i(t)$ with the neurone output $y(t)$ is

$$\Delta w_i(t) = \mu x_i(t) y(t)$$

where

$\Delta w_i(t)$ *represents the temporal change of* $w_i(t)$, *the synaptic weight and*

μ *is the learning rate.*

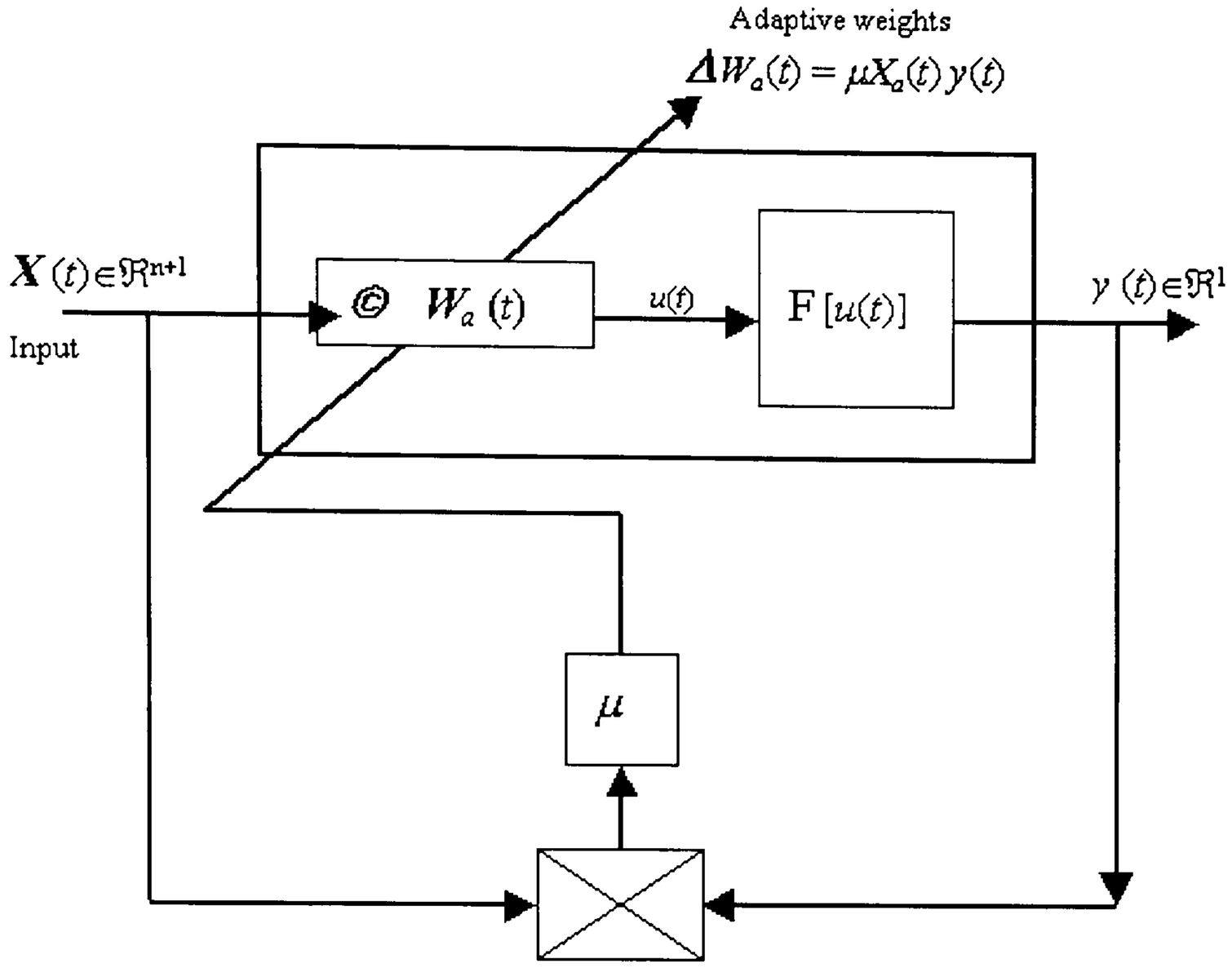


Figure 4.8
An output-based (unsupervised) learning scheme, called Hebbian learning

Competitive learning algorithms are applied to neural layers that contain extensive inter-neurone connections. Competitive learning is based on the notion of "winner take all." In its simplest form, when an input pattern is presented to the neural layer, each neurone competes with all others by transmitting a positive signal to itself using self-excitatory recurrent connections, and sending negative signals to all its neighbouring neurones via lateral inhibitory (competitive) connections. After a period of time, the neurone with the greatest activation state will remain active, called the winner, and all others will be nullified.

5 ARTIFICIAL NEURAL NETWORK PARADIGMS

5.1 MULTI-LAYER FEED-FORWARD MODELS

A single neurone can perform certain simple pattern-detection functions, but, the power of neural computation comes from the number of processing elements connected in a network structure. Larger networks, with their processing elements arranged in layers, generally offer greater computational capabilities. The multi-layer feed-forward model, is one of the most popular artificial neural network architecture. It consists of an input layer, a number of hidden layers (typically one or two) and an output layer. Data flows through the network in one direction only, from input to output. This network can approximate non-linear functions to any desired degree of accuracy. A simplified block-diagram representation of a typical multi-layer static neural networks (MNN) consisting of an input layer, an output layer, and one hidden layer of artificial neurones, with the input vector $X(t) \in \Re^n$ and output vector $Y(t) \in \Re^m$ is given in Figure 5.1. Such structures are inherently stable because they have no feedback. During learning, however, externally imposed feedback interaction takes place. Once trained, each application of a given input set always produces the same output set. Examples of MNN include the multi layer perceptron and radial basis function 'RBF' network.

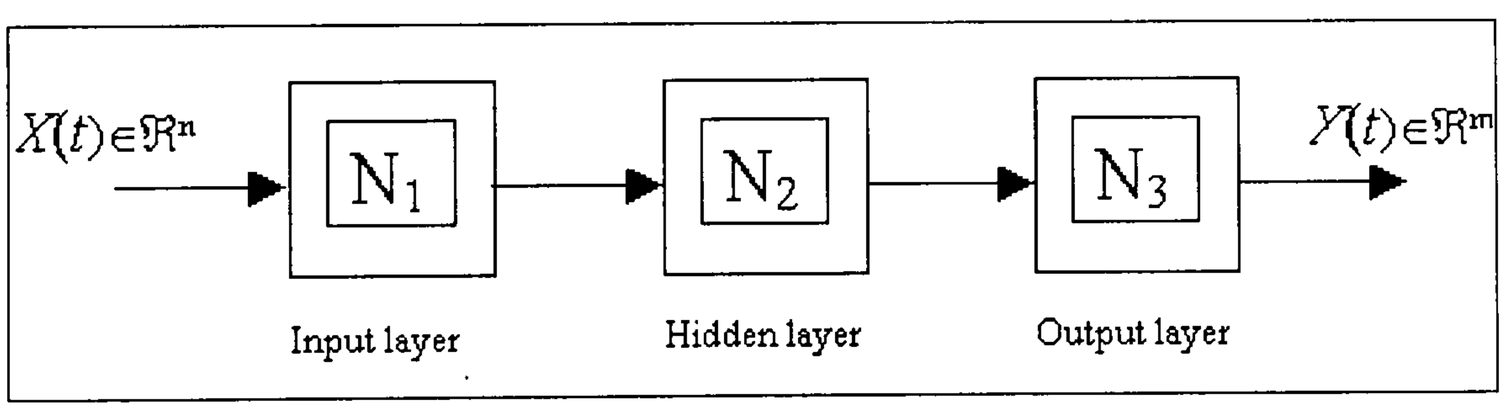


Figure 5.1

- The multi-layer neural network (MNN) with an error back-propagation (BP) algorithm is the most commonly used artificial neural network paradigm in pattern recognition and classification, signal processing, as well as image processing and vision. However, they suffer from many limitations (Hopfield, 1990) including:

- the very slow learning rate- attempts to accelerate the learning process by increasing the values of the learning algorithms' gains [constants], generally result in unstable systems,

- having no dynamic memory- their response depends solely on the current inputs and the values of the weights, many practical questions remain unanswered- for example the number of middle layer processing elements, or the relationship between the accuracy of the function being approximated with the number of hidden layers.

- RBF, Radial basis function networks, have non of back-propagation training problems. Training is usually orders of magnitude faster than back-propagation. RBF feed-forward multi-layer structures are universal approximators (Girosi et al., 1991). They can approximate any continuous function with arbitrary accuracy.

Their main disadvantages are requiring all, or a substantial portion, of the training set to be involved in their operation, and slow operation after completion of the rapid, reliable training. This is due to the large number of variables involved.

Similar and closely related networks are referred to as: 'Gaussian potential function'(Lee and Kil, 1991); 'localised receptive fields'(Moody and Darken, 1988); 'regularisation networks'(Poggio and Girosi, 1990); and 'locally tuned processing units'(Moody and Darken, 1989).

5.1.1 Mathematical representation

Mathematically, the input-output mapping of the MNN shown in Figure 5.1, can be represented by

$$Y(t) = N_3 [N_2 [N_1[X(t) \in \Re^n]]] \in \Re^m$$

In terms of the confluence and non-linear activation operators, this can be rewritten as

$$Y(t) = F^3[W_a^3(t) \copyright F^2[W_a^2(t) \copyright F^1 [W_a^1(t) \copyright X_a(t)]]]$$

Where

$F^i[\cdot]$ *is tile non-linear activation operator,*

$\copyright$ *is the confluence operator* (*scalar product or distance measure*), *and*

$W_a^1(t)$,

$W_a^2(t)$ *and*

$W_a^3(t)$ are the augmented synaptic weight vectors for the input, hidden, and output layers, respectively.

During the learning process, the elements of the synaptic matrices $W_a^1(t)$, $W_a^2(t)$ *and* $W_a^3(t)$ are continuously updated to the new information. Supervised learning algorithms based on an error-correction procedure are often used to determine $\Delta W_a^1(t)$, $\Delta W_a^2(t)$ *and* $\Delta W_a^3(t)$. One method to adapt the feed-forward weights of the processing elements in a static neural network is to minimize the least-mean square (LMS) error (Widrow and Lehr, 1990) between the computed and desired outputs for each neuron in the network. The feed-forward connection of static networks can also be updated using a gradient descent error correction algorithm that is commonly called 'back-propagation' (Hammerstrom, 1993 a and 1993 b).

5.1.2 Back-propagation

For multi-layer static networks, the most popular learning rule in use is the back-propagation algorithm. Back-propagation is a generalisation of the least-squares rule for a multi-layer neural network. It attempts to reduce the error at each neural node in such a way that it minimises the distribution of the weights and improves the information content previously encoded in the weights. The error at each output node is easily determined knowing the target output for each neural node. However, for the hidden layers where outputs are internal to the network and have no explicit target output, the error calculation is much more difficult. The error at a hidden node is literally defined as the amount that its own output is responsible for the error in each neurone in the adjacent layer. It is these hidden layers that serve as abstract domains, into which inputs are mapped. These hidden layers emphasise the differences and de-emphasise similarities between inputs to allow the network to differentiate between trajectories with only subtle differences. Back-propagation can be applied to networks with any numbers of hidden layers by first calculating output, determining the output error, then recursively propagating the error backwards to each layer and adapting the weights to minimise the error. The principle of the back-propagation learning algorithm may be summarised as follows (Hammerstrom, 1993 a and 1993 b).

1. A typical BP nn structure consists of input, hidden, and output layers. Hidden layers may have more than one layer. It is not very clear in the neural network paradigm how many hidden layers are necessary for a particular application. There is not much computation taking place at the input layer. The number of neurones in the input layer equals the

number of input vector components. During learning (training), the input layer sends input information to all hidden nodes.

2. The hidden neurones broadcast their results to all output neurones. Each output neurone calculates a weighted sum and subtracts its actual results (actual response) from its desired, results (targeted output) to produce the error vector (output error).

3. The output nodes calculate the partial derivatives of error-vector components with respect to the weights, and pass these derivatives back to the hidden layer. This computation during learning gives the algorithm its name: the back-propagation. Each hidden neurone calculates the sum of the error derivatives to find its contribution to the output error. Each neurone in the hidden and output layers changes its weight according to a predetermined rule as described by

$$w_i(t+1) = w_i(t) + \Delta w_i(t)$$

where

$$\Delta w_i(t) = \mu x_i(t)\left[y_d(t) - y(t)\right]$$

and

w_i *is the synaptic weight corresponding to*

$x_i(t)$, *the input. The parameter*

$\Delta w_i(t)$ *is the change in*

$w_i(t)$ *synaptic connection over an instant in time,*

μ *is the learning rate,*

$y_d(t)$ *is the desired neural output, and*

$y(t)$ *is the actual neural response.*

In summary, the back-propagation algorithm requires a desired response during learning to compute the error signal, by subtracting the actual outputs (signals obtained from the output neurones) from the desired signals, and then adjusts the weights. After this initial learning, the network could be presented with a new set of data that was not used during the learning

process. The network's accuracy with data outside the learning set gives the generalisation ability to the neural network, id this indicates reliability of the network.

5.1.3 Function approximation

It is well established that feed-forward neural networks can approximate non-linear functions to any degree of accuracy. This ability to approximate non-linear functions is one of the most important attributes of neural networks. The study of function approximation is of primary importance in the neural network paradigm. The intent of this section is to provide the basic concepts of the theory of functional approximation, and to briefly explain the approximation theorems that are normally used to prove that multi-layer static, feed-forward, networks can approximate arbitrary continuous functions to the desired degree of accuracy.

- A function $f(x)$ is defined by real numbers x on a set X if a law prescribed according to which a real number y, y = f (x), is associated with every number x of the set X.

- The aim of function approximation theory is to find a sequence of either algebraic polynomials that converge uniformly to functions continuous in the interval $[a, b]$, or trigonometric polynomials that converge uniformly to a periodic (with period 2π) continuous function. The terms functional and function are used interchangeably in the literature.

- Definition of functional approximation problem

  Given a point g and a set M in a normal linear space S, a point of M of minimum distance from g is called a 'best approximation', and the problem of determining such a point is called a 'best approximation problem'.

  To measure the quality of the approximation, using a distance function $d(F, f)$ to determine the distance of an approximation function $F(W_a, x)$ from the difference function $f(x)$, where $W_a(t)$ is the augmented vector of Synaptic weights. The approximation problem can then be stated as follows.

If $f(x)$ is a continuous function defined on a set S, and $F(W_a, x)$ is an approximating function that depends continuously on x, the approximation problem is to determine the parameter W^*_a, such that

$$d\{F[W^*_a, x], f(x)\} \leq d\{F[W_a, x], f(x)\}$$

A solution to this problem, if it exists, is said to be a best approximation. The following points are of primary importance in a functional approximation problem (Watson, 1980):

- The existence of the best approximation
- The uniqueness of the best approximation
- The characterisation of the best approximation
- The construction of methods for determining the best approximations

- **Basic theorems of functional approximation**

In the 1980s and 1990s, many researchers proved that multi-layer static (feed-forward) neural networks can approximate arbitrary continuous functions (Poggio and Girosi, 1990) to the desired degree of accuracy. Either the Stone-Weierstrass theorem or the Kolmogorov theorem has been employed for the theoretical development of functional approximation capabilities of neural networks.

5.2 FEEDBACK MODELS

In the preceding section, a description of multi-layer feed-forward network structures, the structures without any feedback, was presented. This class of neural networks is called static, feed-forward, or non-recurrent networks. Such networks have no dynamic memory as the response of the network depends on its current inputs and the values of synaptic weights. Such networks, since they do not have any feedback, are inherently stable.

In this Section, structures with feedback connections are described. Such networks are known as feedback, dynamic, or recurrent structures. The feedback causes the network to have local memory characteristics. These networks provide some robust computing characteristics, and their dynamics provide greater insights into biological structures. Recurrent networks are particularly appropriate for modelling [identification] and filtering applications. However, their general theory with regard to architecture and learning algorithms has yet to be developed. The development of better feedback models could result in significant progress in neural network applications.

Because feedback neural networks have feedback paths from their outputs to the inputs, the response of such networks is 'dynamic' or 'recursive'. That is, after applying a new input, the output is calculated and fed back to modify the input. The output is then recalculated, and the process is repeated. For a stable network, successive iterations produce smaller and smaller output changes until eventually the outputs become constant. Under some situations, the process may never end, and such networks are said to be unstable. Unstable networks have interesting properties, and one example of such a network is the 'chaotic system'.

Neural networks with feedback offer great computational advantages over feed-forward networks. For example, it is well known that an infinite-order FIR (finite impulse response) filter, which is only a feed-forward network, is equivalent to a single-pole IIR (infinite impulse response) filter, shown in Figure 5.2.

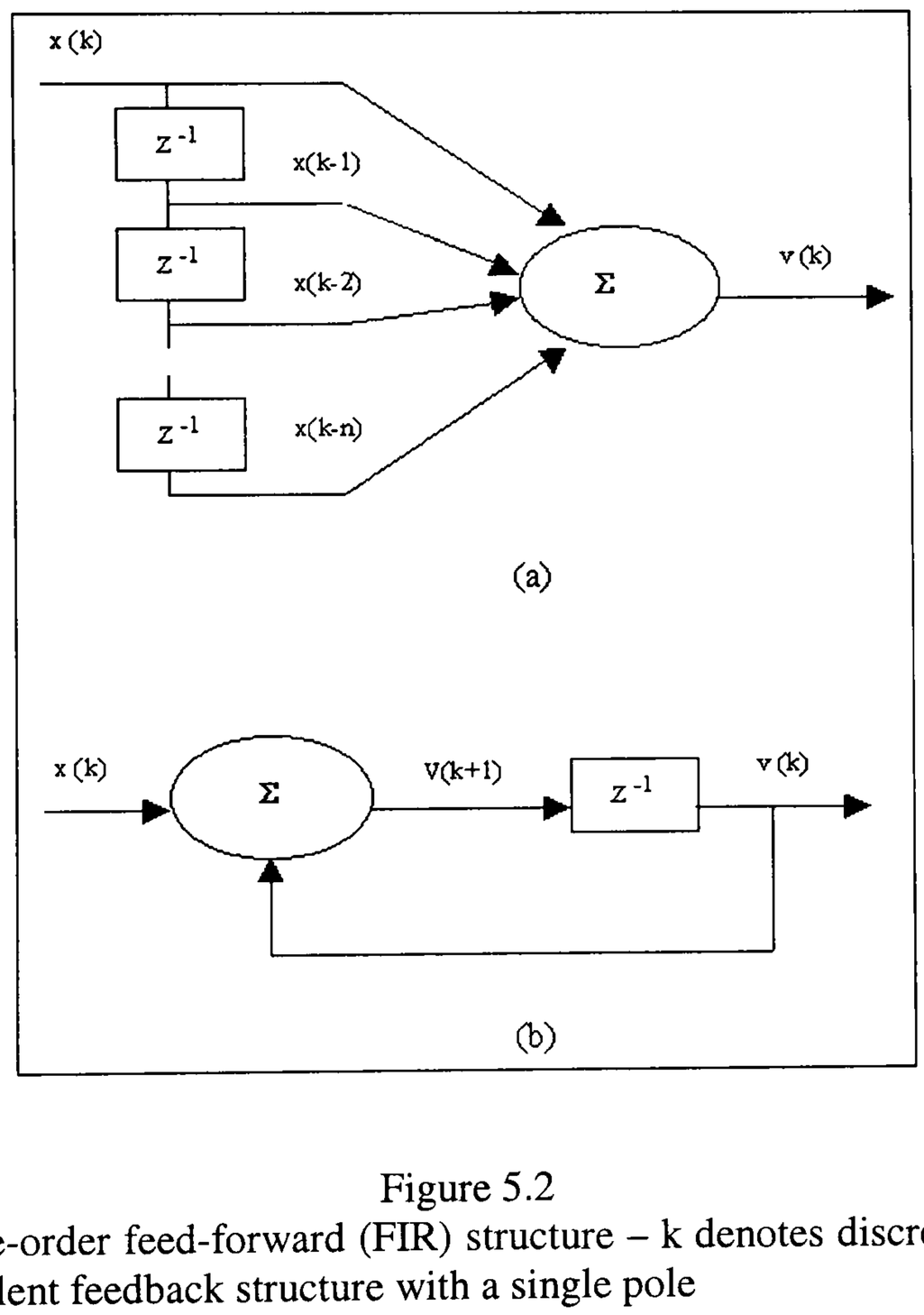


Figure 5.2
(a) An infinite-order feed-forward (FIR) structure – k denotes discrete time index.
(b) An equivalent feedback structure with a single pole

From Figure 5.2 (a), the response of a FIR filter with *n* delay lines may be written as

$$v(k) = x(k) + x(k-1) + x(k-2) + \ldots$$

$$+ x(k-n) = \sum_{i=0}^{n} x(k-i),\ n \to \infty$$

(*k denotes discrete time index*)

or alternatively, in transfer function form

$$\frac{V(z)}{X(z)} = 1 + z^{-1} + z^{-2} + \ldots + z^{-n},\ n \to \infty,\ |z| < 1$$

The difference, or recursive, equation that describes the behaviour of a first-order FIR structure, shown in Figure 5.2(b), may be written as

$$v(k+1) = x(k) + v(k)$$

or alternatively, in transfer function form

$$\frac{V(z)}{X(z)} = \frac{1}{1-z^{-1}} = 1 + z^{-1} + z^{-2} + \ldots + z^{-n},\ n \to \infty,\ |z| < 1$$

From the two transfer function forms, it is clear that the two structures are functionally equivalent. From a computational viewpoint, a network with feedback is equivalent to a large, or possibly an infinite, feed-forward structure.

5.2.1 Processing time series data

It is possible to use a static network to process time series data by simply converting the temporal sequence pattern by unfolding the sequence over time. That is, time is treated as another dimension in the problem. From a practical viewpoint, it is possible to unfold the sequence over a finite period of time. This can be accomplished by feeding the input sequence into a tapped delay line and then into a static neural network architecture. An architecture like this is often referred to as a time-delay neural network (TDNN) (Waibel, et al., 1989). This neural structure is basically a feed-forward network with dynamic (delay) elements. Because of the delay operators, the TDNN is categorised under dynamic networks. This is equivalent to a FIR filter whose output forms an argument to a non-linear function (Figure 5.3).

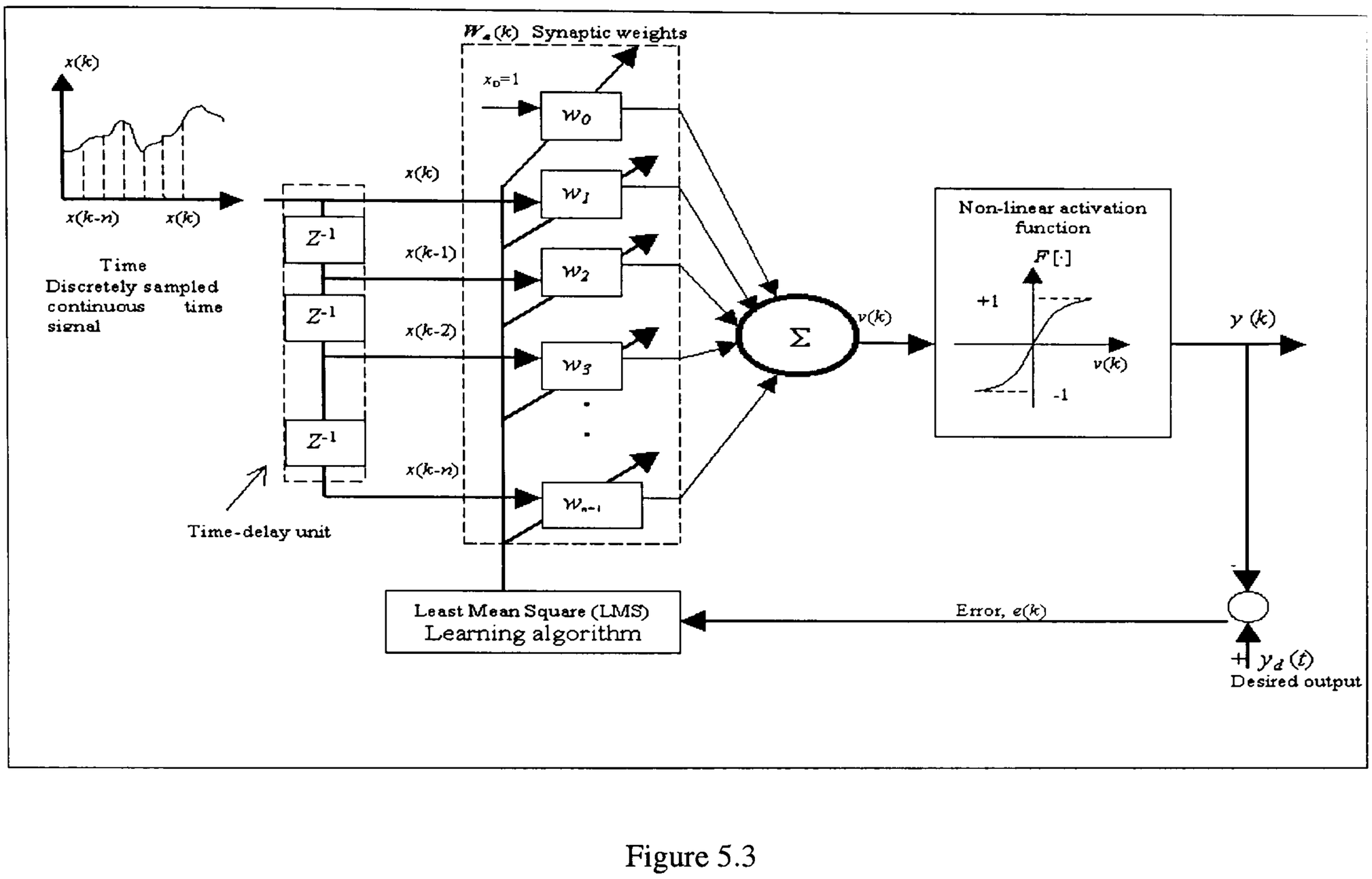


Figure 5.3

The basic time-delay neural unit shown in Figure 5.3 can function as an adaptive filter by computing the scalar product of the augmented input vector $X_a(k)$ and augmented synaptic weight vector $W_a(k)$. The elements of the synaptic weight vector $W_a(k)$ are modified by the least-mean square (LMS) learning algorithm .

The dynamics of a time-delay neural unit are described by the following equations:

$$v(k) = \sum_{i=0}^{n} w_i x(k-i)$$

$$y(k) = F[v(k)]$$

This neural structure has been used in many applications e.g., text-to-speech conversion (Sejnowski and Rosenberg, 1986); phoneme recognition (Waibel et al., 1989).

5.3 NEURO-FUZZY STRUCTURES

5.3.1 Introduction

- Fuzzy logic is a relatively fast and cheap way to map an input space to an output space. The primary mechanism for doing this is a list of if-then statements called rules.

- Fuzzy logic is useful where a problem can be described linguistically or, where there is data and one is looking for relationships or patterns within that data. A fuzzy logic system that accepts imprecise data and vague statements, such as 'good', 'excellent', 'poor', 'generous', and provides decisions is depicted in Figure 5.4.

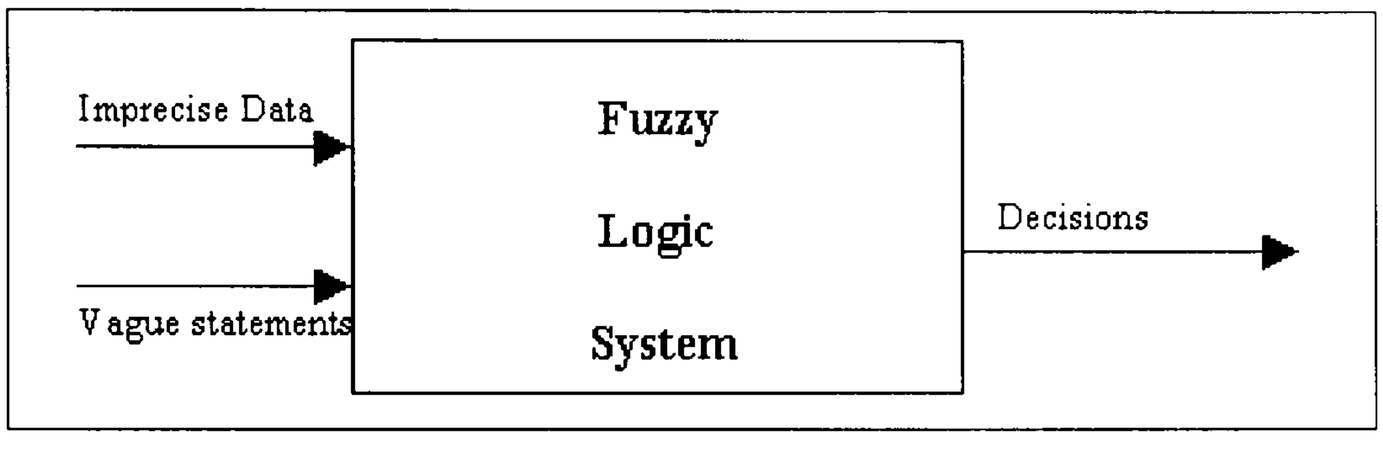


Figure 5.4

- Artificial neural network structures can deal with imprecise data and ill defined activities. However, the subjective phenomena such as reasoning and perceptions are often regarded as being in the domain of fuzzy logic theory. Fuzzy logic provides an inference morphology that enables approximate biological reasoning capabilities.

- Fuzzy inference is a method that interprets the values in the input vector and, based on some set of if-then rules, assigns values to the output vector.

- Reasoning in fuzzy logic is just a matter of generalising the yes-no ,Boolean, logic.

- The neural network approach blends well with fuzzy logic (Jang et al., 1997) and some research endeavours have given birth to 'fuzzy neurones' and 'fuzzy-neural systems'.

- A fuzzy neurone has the ability to cope with fuzzy information. Inputs to the fuzzy processing elements are fuzzy sets

  $(x_1, x_2, \ldots, x_N)$ *in the universe of discourse*

  $(X_1, X_2, \ldots, X_N)$ *respectively.*

These fuzzy sets may be labelled by such linguistic terms as 'high', 'large', 'warm', 'medium', etc. The inputs are then 'weighted' in much different ways from those used in a non-fuzzy case. The weighted inputs are then aggregated not by the summation but by the fuzzy aggregation operations (fuzzy union, weighted mean, or intersection).

- In its simplest form, a fuzzy neurone is designed to function in much the same way as a non-fuzzy neurone. The mathematical operations involved in a non-fuzzy computational processing element, as described earlier, are:

(i) the scalar product between the neural inputs and the synaptic weights, and

(ii) the summation of these products.

$$y(t) = F\left[\sum_{i=0}^{n} w_i\, x_i\right]$$

where

$[x_1, \ldots, x_n]$ *represent neural inputs,*

$[w_1, \ldots, w_n]$ *the synaptic weights,*

$y(t)$ *the neural output and*

$F[\cdot]$ *is some non-linear activation function.*

Modifying the scalar product by fuzzy multiplication and the summation operation by fuzzy addition, lead to a fuzzy-neurone based on fuzzy arithmetic operations. The function of such a fuzzy processing element can be modelled by the following equation

$$y(t) = F\left[\left(\underset{i=0}{\overset{n}{+}}\right) w_i(\cdot)\, x_i\right]$$

where

(+) *and* (·) *are fuzzy addition and fuzzy multiplication operators respectively.*

- Fuzzy-neural systems, still in their infancy, but have already made an impact in the areas of pattern recognition, control systems, system modelling and expert systems (Lawrence and Harris, 1993).

- The computational process envisioned for advanced neuro-fuzzy computational systems is as follows. It starts with the development of a 'fuzzy processing element' based on the understanding of biological neuronal morphologies, followed by learning mechanisms. This leads to the following three steps in a neuro-fuzzy computational process:

  (i) development of fuzzy neural models motivated by biological neurones,

  (ii) models of synaptic connections that incorporates "fuzziness" into artificial neural network, and

  (iii) development of learning algorithms (i.e., the method of adjusting the synaptic weights).

5.3.2 Fuzzy sets

- In fuzzy logic the truth of any statement becomes a matter of degree. Fuzzy logic is based on fuzzy sets. The concept of fuzzy sets was introduced by Lotfi Zadeh in 1965 (Lotfi-Zadeh, 1992) in order to represent and manipulate data that were not precise, but were, instead, "fuzzy". The fuzzy set theory provides a mechanism for representing linguistic constructs and vague concepts.

- A fuzzy set can be defined as a set without a crisp, clearly defined boundary. It can contain elements with only a partial degree of membership. The degree an object belongs to a fuzzy set is denoted by a membership value (0 to 1).

- A membership function associated with a given fuzzy set maps an input value to its appropriate membership value.

- **Definition**

If X is the universe of discourse, a space of points (or objects), with its elements denoted by x,

then for a classical set, say of real numbers > 6

$$A = \{ x \in X \mid x > 6 \}$$

If $x > 6$, *then x belongs to the set A, otherwise x does not belong to the set A*

A fuzzy set is an extension of a classical set. A fuzzy set A is a subset of the universe of discourse X that admits partial membership. It is characterised by a membership (characteristic) function $\mu_A(x)$. The fuzzy set A in X is defined as an ordered pair

$$A = \{ x, \mu_A(x) \mid x \in X \}$$

Where

$$0 \leq \mu_A(x) \leq 1.$$

$\mu_A(x)$, *the membership function, describes the degree to which the object x belongs to the set A.* $\mu_A(x)$ *is also referred to as the 'characteristic function' or 'graded membership' of x in A. If*

$\mu_A(x)=0$ *then it is certain that x is not in A, and*

$\mu_A(x)=1$ *then it is certain that x is in A.*

For x with $0 < \mu_A(x) < 1$, *there is an uncertainty associated with x, that is, x belongs to A with the possibility* $\mu_A(x)$.

- **Architectures**

Artificial neural networks and fuzzy logic integrate very well and architectures based upon this integration are believed to have considerable potential in the areas of pattern recognition, system modelling, control systems, and expert systems medical diagnosis. Two possible models of fuzzy neural systems are schematically shown in Figures 5.5 (a) and 5.5 (b).

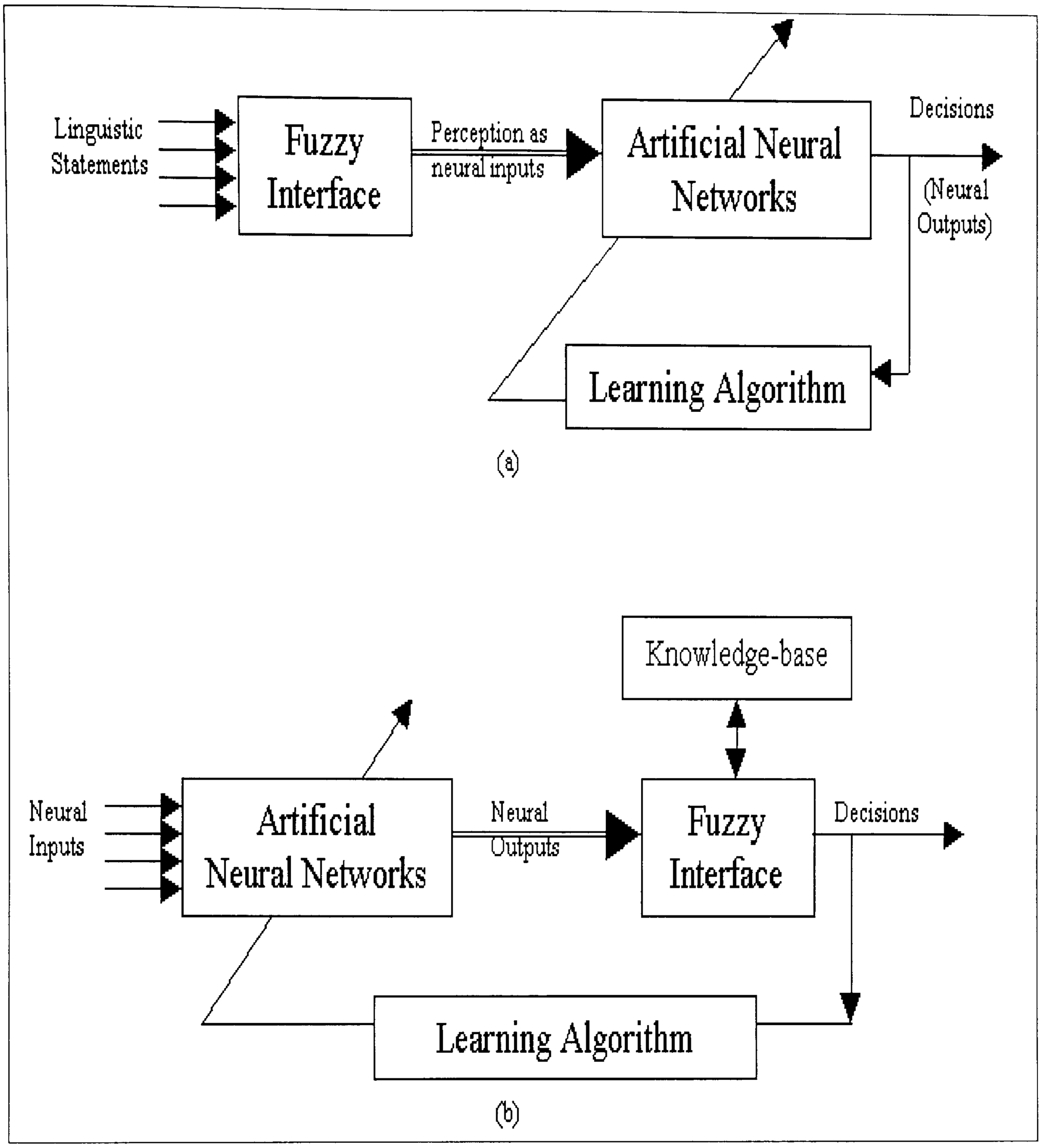


Figure 5.5

The area of neural structures in general, and fuzzy neural structures in particular, is a fertile area of theoretical and applied research. This Chapter presented a brief description of various structures. In the next Chapter, links between artificial neural networks and fractals are investigated.

6 ARTIFICIAL NEURAL NETWORKS AND DYNAMIC SYSTEMS THEORY

6.1 OVERVIEW

The theory of neural networks is not yet sufficiently matured to allow an artificial neural network application, using fractal noise, to be precisely designed and developed. The process involves trial-and-error as well as research and planning. Carefully controlled experimentation alone may be sufficient in some cases, but, general mathematical principles or new mathematical theories are required to support experimental discoveries. Using some general mathematical principles and results developed by Felix Hausdorff and Stefan Banach, the purpose of this Chapter is to explore any links between ANNs and fractals. After an introduction to iterated function systems, the connection between ANNs, fixed point theory, and chaos is covered

6.2 DYNAMICAL SYSTEMS

A good metaphor for a dynamical system with one attractor is the ball in the bowl experiment in which a small metal ball put into different positions of a bowl and then let loose, always comes to rest at the bottom of the bowl. If the ball is put at the bottom, nothing happens. In a situation with two or more attractors, the final development depends on other issues. The field of dynamical systems theory addresses such issues and provides the framework for discussing chaos and the generation of fractals. Chaos resides in the realm of non-linear dynamical systems. They involve a non-linear process and feedback.

e.g.

a discrete dynamical system

$$x_{n+1} = f(x_n)$$

generates a trajectory, or orbit, of the form

$$x_0, x_1, x_2, x_3, \ldots$$

where

-the output of one operation is the input for the next iteration

-f is a non-linear function mapping, e.g., $R^n \rightarrow R^n$, *i.e., with the same dimension for input and output variables.*

-x_0 *is some arbitrary starting point*

System behaviour is characterised in terms of the trajectory

1. x_n's $\rightarrow \infty$

2. x_n's, *after some value of n, all equal the same point* x^*,

   i.e. a fixed point of f.

3. *In the case of a periodic P orbit, for any positive integer P, e.g. in a period two orbit, two points repeat:*

   $$x^*, y^*, x^*, y^*, \ldots$$

   $$x^* = f(y^*) \quad and \quad y^* = f(x^*)$$

4. x_n *remains bounded, but never repeat, i.e., once an* x_n *repeats a previous value, the orbit is locked into periodicity. The orbit points lie on a bounded attracting set called a strange attractor – i.e. the signature of chaos.*

   An example of dynamical systems producing strange attractors is Henon mapping, generated by the mapping

   $f_{a,b}: R^2 \rightarrow R^2$ *defined by*

   $$f_{a,b}(x_n, y_n) = (1+y-ax^2, bx)$$

   The dynamical system is given by

   $$(x_{n+1}, y_{n+1}) = f_{a,b}(x, y)$$

This represents a family of dynamical systems on R^2 *parameterised by real parameters a and b, the value of which determine the nature of the dynamical system.*

The values a=1.4 and b=0.3 produce a strange attractor and

different values of a and b, produce a variety of different behaviours, including fixed points, periodic orbits, and other strange attractors.

6.3 ITERATED FUNCTION SYSTEMS

6.3.1 A metaphor to describe an IFS

- The ball in the bowl experiment, also, corresponds to a common metaphor to describe an iterated function system (IFS): the multiple reduction contractive photocopier. Using a number of lens systems, to reduce the size of an input image independently, the photocopier paste together reduced copies of the original image to form the output. The output is then fed back as its new input repeatedly. Regardless of the initial input, the output sequences of images always tends towards the same final result, called the invariant attractor of the machine. Invariant because using the photocopier the output is equal to the input.

- In comparison to the ball in the bowl experiment, the copy machine here corresponds to the bowl, and the output sequences of images toward the invariant attractor corresponds to observing the path of the ball to the rest point. Using results developed by Felix Hausdorff and Stefan Banach, it has been shown that (Hutchinson, 1981) any multiple reduced photocopier, in which the number and the design of the lens system may change and each lens system contracts images, always has a unique final attractor.

- An IFS consists of a collection of contractive transformations which maps the plane R^2 to itself.

$$\{w_i : R^2 \to R^2 \mid i = 1,...,n\}$$

For a given initial set *S*, with a set of affine transformations (i.e. the lens systems of the photocopier)

$$w_1, w_2, w_3,...$$

small affine copies $w_i(S)$ are produced for each *I*

The collection of transformations, or the assembly of the reduced copies, defines a map *W* that is applied to sets and a new set $W(S)$ is obtained which is the equivalent of the output of the photocopier.

$$W(\cdot) = \bigcup_{i=1}^{n} w_i(\cdot)$$

W is called the Hutchinson operator. Running the photocopier in feedback mode thus corresponds to iterating the operator W. The Hutchinson's operator turns the repeated application of the metaphoric photocopier into a dynamical system. This is the essence of a deterministic iterated function system (IFS).

- Hutchinson (Hutchinson, 1981) showed that the operator W, which describes the collage is a contraction with respect to the Hausdorff distance.

  i.e., there is a constant c, with $0 \leq c < 1$, *such that*

  $$h(W(A), W(B)) \leq c.\, h(A, B)$$

  for all (compact) sets A and B in the plane.

In establishing this fundamental property, Hutchinson brought into consideration The Contraction Mapping principles. which owes its final formulation to Stefan Banach (1892-1945) (Hutchinson, 1981).

6.3.2 The contractive mapping fixed point theorem

Given a contractive map W on a space of images, there is a special image x^* with the following properties:

1) x^* is the fixed point of W. This corresponds to the output of the photocopier being equal to its input.

$$W(x^*) = x^*$$
$$= w_1(x^*) \cup w_2(x^*) \cup \ldots w_n(x^*)$$

2) x^* is the limit set, and it is not dependent on the choice of initial image

Starting with some initial image S_0,

$$S_1 = W(S_0),$$

$$S_2 = W(S_1) = W(W(S_0)) \equiv W^{\circ 2}(S_0)\ ,\ \textit{i.e. the output of the second iteration}$$

where ∘ *indicates iteration*

An IFS generates a sequence which tends towards a final set S_∞, or the attractor of the IFS, and which is not dependent on the choice of S_0.

$$x^* \equiv S_\infty = \lim_{n \to \infty} W^{\circ n}(S_0)$$

3) x^* is unique. In terms of S and W this means that if any set S satisfies

$$S = W(S)$$

then S is the attractor of W

$S = x^*$ *i.e. only one set will satisfy the fixed point equation*

$$W(x^*) = x^* = w_1(x^*) \cup w_2(x^*) \cup \ldots w_n(x^*)$$

6.3.3 The Hausdorff metric

The Hausdorff metric determines the distance between points of a space X, measured by a function d: $X \times X \rightarrow$ R, where R denotes the real numbers. To find the Hausdorff distance $h(A, B)$ between two subsets of the plane, A and B, for each point x in A, the closest point y in B is found. These minimal distances are measured and the largest one is the Hausdorff distance.

Mathematically, given a complete metric space (X, d), the Hausdorff metric h, is defined as follows:

$$h(A, B) = max\ \{d(A, B), d(B, A)\}, \quad A\ and\ B \in H(X),$$

where

$H(X)$ *is the Hausdorff space whose points are the compact subsets of* X

$$d(A, B) = max\ \{d(x, B) : x \in A\}$$

$$d(x, B) = min\ \{d(x, y) : y \in B\}$$

d *is the standard distance function*

The function d must have the properties that

1) $d(x, y) \geq 0$

2) $d(x, y) = 0$ *if and only if* $x = y$

3) $d(x, y) = d(y, x)$

4) $d(x, y) \leq d(x, z) + d(z, y)$ *triangle inequality, hold for all* $x, y, z \in X$

Examples of a metric space are

1) *For real numbers x and y*

$$d(x, y) = |x - y|$$

2) *For points* $P = (x, y)$, $Q = (u, v)$ *in the plane it can be defined*

$$d_2(P, Q) = ((x-u)^2 + (y-v)^2)^{1/2}$$ *-the Euclidean metric*

6.3.4 The inverse problem

Given a set S, find an IFS whose attractor is S. No satisfying solutions exist, to this problem, but, some insight can be found in the equations of the fixed point and the collage.

The fixed point equation states that the fixed point x^* is constructed out of transformed copies of itself:

$$x^* = W(x^*) = w_1(x^*) \cup w_2(x^*) \cup \ldots w_n(x^*)$$

That is, given the set S, transform it by contractive transformation and paste these together to reconstruct S. The uniqueness of x^* is important because,

if given either S or W, the other can be found respectively, satisfying

$$S = W(S),$$

Then S is the attractor of W

$$S = x^*$$

The collage theorem declare that even if the pieces can't be pasted to fit exactly, the better the fit between the original set S and the pasted 'collage' $W(S)$, the closer x^* will be to S (using transformations that are barely contractive).

$$d(S, x^*) \leq \frac{1}{1-s} d(S, W(S))$$

s close to 1 *for barely contractive transformations*

6.4 ARTIFICIAL NEURAL NETWORKS AND FIXED POINTS

An artificial neural network has many forms. However, mathematically, the dynamics of all types of neural network can be summarised as discovery of fixed points in the system and contraction toward the established fixed point. This section summarises the link between artificial neural network and fixed points.

Considering the typical multi-layer hierarchical neural network previously shown in Figure 5.1, the output function of individual processing elements which have several input and output can be expressed as

$$f(x) = \sigma\left(\omega_0 + \omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_n x_n\right)$$
$$\sigma(u) = \frac{1}{1+\exp(-u)}$$

$(x_{0,\ldots,} x_n)$ *are n input values of the sequence of real number.*

x is within in the open interval (0,1)

$(\omega_{0,\ldots,} \omega_n)$ *another sequence of real number and constitutes the weights of processing elements.*

For the hidden layer processing elements j = 1, ..., m. each processing element has several input values. Their respective output value

$$y_i = f_j(x) = \sigma\left(\omega_{0j} + \omega_{1j} x_{1j} + \omega_{2j} x_{2j} + \cdots + \omega_{nj} x_{nj}\right)$$

become the input of the next layer. Thus. of neurone k is also determined.

$$z_k = g_k(x) = \sigma\left(v_{0k} + v_{1k} x_{1k} + v_{2k} x_{2k} + \cdots + v_{nk} x_{nk}\right)$$

This input-output relationship can be rewritten into a general combined equation

$$h(x) = g\left(f_1(x), \cdots, f_m(x)\right)$$

So far, given the elements x of the training pattern of the finite set $S \subseteq X$ and given the value d of the objective function $d(x)$, the training of hierarchical neural network is actually a function interpolation, solving $h: X \to (0,1)$.

Normally, after solving function h of the training pattern and function d of the objective model, the weights ω_{ij} and v_{jk} can be determined when there is a minimal solution of

$$E = \frac{1}{|S|}\sum_{x \in S}\left[h(x) - d(x)\right]^2$$

which is the mean square error.

The solution of the minimum value of such non-linear function normally uses a gradient method. When ω_{ij}, v_{jk} is a function of time t, the gradient system of error E is expressed in the following:

$$\frac{d\omega_{ij}}{dt} = -\frac{\partial E}{\partial \omega_{ij}} \quad \frac{dv_{jk}}{dt} = -\frac{\partial E}{\partial v_{jk}}$$

If ω_{ij}, v_{jk} is the solution to the above differential equation, then

$$\frac{dE\left(\omega_{ij}(t)\ v_{jk}(t)\right)}{dt} \le 0$$

The condition that satisfies this is

$$\frac{\partial E}{\partial \omega_{ij}} = 0 \qquad \frac{\partial E}{\partial v_{jk}} = 0$$

Thus, assuming δ is a very small positive constant number,

the sequence $\omega_{ij}(s\delta)$, $v_{jk}(s\delta)$ (s =1,2, 3,....) created by the slope function

$$\omega_{ij}(t+\delta) = \omega_{ij} - \delta\frac{\partial E}{\partial \omega_{ij}}$$

$$v_{jk}(t+\delta) = v_{jk} - \delta\frac{\partial E}{\partial v_{jk}}$$

which starts from the appropriate initial value $\omega_{ij}(t_0)$, $v_{jk}(t_0)$ will enable error E to contract toward the minimal point in the system (normally the relative minimum point).

If error E is also viewed as a function of ω, the training process of

$$\omega_{ij}(t+\delta) = \omega_{ij} - \delta \frac{\partial E}{\partial \omega_{ij}}$$

$$v_{jk}(t+\delta) = v_{jk} - \delta \frac{\partial E}{\partial v_{jk}}$$

is actually a self-mapping in the N dimensional vector space.

Many repetitions of the above can obtain

$$(grad\ 2,\ E)(\omega) = \left(\frac{\partial E}{\partial x_1}(\omega), \cdots, \frac{\partial E}{\partial x_N}(\omega) \right).$$

If

$$F: \omega \to \omega - \delta(grad\, E)(\omega)$$

satisfies the complete conditions of fixed point contraction

$$f(x_{n+1}) = k\, x_n - s(x_n) g(x_n) - h(x_n)$$

it can be rewritten as

$$\omega_{n+1} = k\omega_n - s(\omega_n) g(\omega_n) - h(\omega_n).$$

Here,

$$k = 1, \quad s(\omega_n) = \delta, \qquad h(\omega_n) = 0.$$

From the above we know that the weight ω obtained as a result of training is actually the fixed point in the map F, i.e. the process of training or self-organisation is actually a dynamics which increasingly moves toward the set fixed point through constantly upgrading itself.

Mathematically, the dynamics of all types of neural network can be summarised as exploration of fixed points in the system and contraction toward the established fixed point.

6.5 SUMMARY

A copy machine was considered, which was essentially an arrangement of lens systems which contract images, generating a dynamical system (an IFS). Running the machine in a feedback environment leads to a sequence of images A_0, A_1, A_2, ... A_∞, where A_0 is an arbitrary initial image. The final image, A_∞, is independent of A_0. If we chose A_∞ as the initial image, then nothing happens. That is, the IFS leaves A_∞ invariant. It is said that A_∞ is a fixed point of the IFS, or that A_∞ is an attractor for the dynamical system. In this sense, the resulting attractor can be identified with IFS.

With regards to artificial neural networks, as mentioned above,

- Mathematically, the dynamics of all types of neural network can be summarised as exploration of fixed points in the system and contraction toward the established fixed point.

- In discrete dynamical system, neural network and chaos are closely and extensively linked because of the fixed point.

- The existence of such a link makes it possible to introduce chaos into neural network based on the fixed point theory.

7 PRACTICAL WORK

7.1 OVERVIEW

Since the 1970s many mathematically optimal solutions have been developed for specific tasks in signal processing and identification of objects from observed patterns or images. These solutions break down however, when adapted for similar problems or when the underlying assumptions that may be unreasonable in practice are violated. For example, the common assumption that noise (typically thought of as the enemy of order), follows a particular distribution is known to play a detrimental rule in many experimental situations.

The practical work carried out in this and the following Chapters investigates the applicability of artificial neural networks to real world signal processing and pattern recognition tasks, addressing issues with regard to noise, mapping, and generalisation.

In this Chapter, employing a pattern recognition problem in biochemical process control, important considerations in dealing with noise, incorporation of approximate models and ensuring optimum results are experimentally analysed. Joint publications based on this work, by Professor J. M. Blackledge and the author were presented at the 3rd ISC'97 Rzesow University of Technology, Poland, and in Recent Advances in Soft Computing'98, DMU, respectively. A modified version of these papers are included in Appendix C.

7.2 PROBLEMS

In chemical and biochemical processes a key goal is to manage transitions from one flow state to another in an efficient manner so as to maximise profit over time. Even a half of a percent improvement in efficiency could save the industry in the order of millions of dollars (Ungar, 1990). To address such issues, one must address problems regarding noise, incorporation of appropriate models, lag times, 'credit assignment' (what action was responsible for an observed effect), optimisation over time, and non-linear processes.

A key goal of such strongly non-linear processes is to manage transitions from one flow state to another in an efficient manner so as to maximise profit over time. Although for most of these processes extensive data is available from past runs, it is difficult to formulate precise models. Noise in the form of fluctuations that do not carry information is present in such applications. Many controllers for non-linear systems make the unrealistic assumption that

exact non-linear models of the process being controlled are known. Even those controllers that can include arbitrary non-linear functions of the output and old inputs are typically limited to single input single output systems (Agrawal and Seborg 1987).

7.2.1 A prime example

The bio-reactor is a prime example of a challenging non-linear problem. Improvements in bio-reactor control techniques can result in significant savings to the biochemical industries. Apart from noise, the issues that make it difficult to control include time delay, i.e. lag times between the time a control action is taken and the time a response is observed, non-linearity, and instability. The simplest version of the bio-reactor is a continuous flow stirred tank reactor (CFSTR) containing water, nutrients ('substrate') and biological cells (e.g. bacteria). The state of the process is characterised by the number of cells and the amount of nutrients. In this simple form cell growth depends only on the nutrients being fed to the system. The volume in the tank is maintained at a constant level by removing tank contents at a flow rate equal to the incoming flow rate. The objective is to achieve and maintain a desired cell amount by altering the flow rate. Significant delays exist between changes in flow rate and the response in cell concentrations. Additionally, the problem exhibits multiplicity, i.e. two different values of flow rate can lead to the same desired set-point in cell mass yield. Utilising its equation form developed by Agrawal et al. (1982) for such a bio-reactor, it was shown that small changes in parameter values can cause the bio-reactor to become unstable (Ungar 1990). It should be noted that these tests unrealistically assume that there is no noise in the measurements (e.g. of amount of cells) and that the control parameter (the flow rate) actually takes on the exact value that the controller requires.

To cope with uncertainties regarding plant dynamics and its environment, networks that can learn to compensate for deficiencies in performance of conventional controllers may prove useful. In the following Section, employing a benchmark problem, important considerations with regard to noise and incorporation of approximate models, ensuring optimum results, are experimentally analysed.

7.3 BENCHMARK

7.3.1 Objectives

Using a bio-chemical pattern recognition problem, the objectives of the benchmark were:-

To address issues regarding noise and incorporation of approximate models.

To investigate and evaluate mapping and generalising attributes of various neural paradigms for pattern recognition.

To experimentally analyse the robustness of artificial neural networks to noisy data such as that which occurs in real world applications.

7.3.2 Details

Many of the limitations of conventional solutions arise on relying on poorly modelled non-linear systems, e.g. unrealistically assuming that there is no noise in the measurements and that the variable (e.g. flow rate) takes on the exact value that the controller requires are typical in conventional testing. Using a pattern recognition problem, such issues were practically investigated.

In the benchmark, a large number of patterns consisting of fluid flow readings from a biochemical rig were gathered. Each input training pattern consisted of the fluid flow readings in litres taken at five second intervals over a period of one minute. The patterns could be divided into three broad categories representing the state of the system output, i.e. stable or unstable fluid flow and transition from one flow level to another. The changes to the fluid flow were set both manually and automatically. Various neural paradigms were constructed and used to recognise the three categories of patterns. The outputs of such networks can be used as inputs to a multi input neural controller, enabling the controller to take account of the exact state of the flow rate. The learning and generalising ability of these networks, particularly in a noisy environment, in which conventional systems perform poorly, were systematically examined and compared.

Many processes are dynamic non-linear systems. Recurrent neural networks appear to be ideally suited for modelling non-linear systems. The response of such networks is dynamic or recursive. For a stable network, successive iterations produce smaller and smaller output changes until eventually the outputs become constant. For an unstable network, the process

may never end. Unstable networks have interesting properties, and one example of such a network is the chaotic system. As part of the experiments a number of recurrent back propagation networks were designed and applied to the problem. Their contributions in pattern recognition were then evaluated.

7.3.3 The development environment

- Most network designers, rather than becoming involved in sequential programming, prefer focusing exclusively on the building, training and testing process, using a software development environment with the appropriate tools. They only become involved in sequential programming in the latter stages of a project for fine tuning and optimisation purposes. The approach is necessary, because, for example, in comparison to a hard computing program in which a list of all the variables or instructions can be readily printed and analysed, a list of all the weights and connections in a neuro-computing is not easily understood. However, such development tools have many limitations and tailor-made environments may be required in the latter stages of a project.

- For this research project a combination of C/C^{++} was used to develop the neural paradigms. Ready made networks were also downloaded from the internet. The main tool used for fractal modulation/demodulation was Matlab.

7.3.4 Network design

One way of finding the most appropriate network for the application is to decide upon the training procedure required. In 'the benchmark problem', because the exact output for every input is defined, supervised training is the most suitable. Hence, networks such as multi-layer feed-forward nets using BP are likely candidates. Counter propagation is also a likely candidate because it may use both supervised and unsupervised training.

Various aspects of network design, e.g. deciding upon the number of processing elements or the number of layers, were determined experimentally. That is a large number of networks were built, trained and tested to determine the most successful configuration. This is the most common method used in practice. Details of the experiments are covered in the implementation section.

7.4 DATA

7.4.1 The source

- The sources of training/testing data may include: test data, simulated and hypothesised results, statistical and historical records.
- Training data sets for the benchmark were collected from a bio-chemical process control rig., consisting of :-

  1. Sump, 2. Pump, 3. Analogue/digital controlled diverter valve, 4. Cooler, 5. Process Tank, and 6. Analogue/digital controlled drain valve.

- All features on the unit can be controlled in an analogue or a digital mode. The fluid can be pumped at a flow rate of 0-2 litres / minutes from the pump flowing either directly to the process tank, or first to the cooler and then to the process tank. The fluid cycle is completed by either an overflow or drainage, using either of the two valves provided. Being based around a fluid flow process to control flow and temperature, the rig presents a relatively small but typical operation, analogous to that in the industry.

7.4.2 Training

- A large number of training sets were developed using a maximum of 120 patterns. As will be seen later in the implementation section, with a supervised model such as back propagation, more training examples generally results in a better network response. Each training input pattern represents the fluid flow readings (in litre/minute) taken at 5 second intervals over a period of 1 minute. The input patterns could be divided into three categories representing the output flow states of the system. That is a steady fluid flow (e.g. 1.5 litre/minute), an unstable flow, or a transition from one flow rate to another. The change in the fluid flow was generally set manually, but sometimes also automatically. The sets were then pre-processed before being fed to the system.
- Pre-processing mostly involves scaling down and normalising the data. Scaling down is used to avoid fluctuations in one input with much larger range swamping any importance in a second input with a relatively small range. Normalisation is required when the numeric input data has a natural range that is sometimes other than the processing element's operating range.

- Noise was superimposed on some of the data. Noise in the form of fluctuations that do not carry information is present in real world applications. Hence, superimposing noise on data sets from the rig. accustoms the networks to the noise typically found in an industrial environment. In addition, it provides the network with enhanced data sets. As will be seen in the implementation chapter, another application of noise in neural networks is to help to provide the solution. This is used during training when problems such as stabilising in a local minima may occur. Injecting a small amount of noise helps to provide the solution and can avoid the use of more elaborate techniques such as reinitialising the weights and restarting training or changing the learning parameters.

- Each training output pattern simply consisted of three numbers representing a state (e.g. steady, unstable or changing state). Such outputs could be used as input signals for a multi input plant controller, enabling it to take account of the state of the flow rate. The values for the training output patterns were chosen on the basis of being simple to map to their corresponding input fluid flow patterns.

Appendix A1 provides sample fluid flow patterns of a full training set containing 55 vectors in their digitised as well as the normalised and scaled down versions.

7.4.3 Testing

Networks were tested to evaluate the training process. Important issues for investigation were as follows:-

- Presenting the network with training cases first and then with new cases that were similar but different from the training cases. This is necessary in order to establish that the network has not simply memorised the training input patterns. A well trained network should be able to use its knowledge to recognise new but similar patterns.

- Superimposing noise on data sets and testing the networks on them. Superimposing noise on training data sets accustoms the networks to the noise typically found in an industrial environment, and provides the network with enhanced data sets. Once trained, such a network should be able to correctly identify such patterns.

- Testing on different network configurations (e.g. in terms of variables and processing elements). Because of the lack of theory, the most common method to train a network for a specific application involves many choices and experiments. That is a number

of networks are built, trained and evaluated to determine the most successful configuration.

- Evaluating similar networks trained on a variety of training files. The character of a network is not only determined by its architecture but also by the quality and quantity of the data used. Hence, in order to get the best results a network requires assistance in the form of proper gathering and preparation of its input data. It was found that generally gathering more training data resulted in a more accurate performance by the network, when using a supervised model. With an unsupervised model, too many examples may interfere with each other, resulting in the network's recall ability diminishing.

- The role played by the middle layer processing elements, is another important consideration. When the test set patterns are presented, some processing elements strongly react to a particular pattern. If the same processing element does not react strongly to any other training or testing pattern, then there is a possibility that this processing element is the feature detector for that feature. In practice, however, it was found that the features detected are not the clear-cut features that one might predict. Instead they are usually a combination of obvious and obscure features. An inspection of the weights for the connections coming into a middle layer processing element, for the fairly strong positive values, can assist the developer to guess for the correct features that the processing element scans for. Input connections having strong negative weights are also considered as a feature detector that may be checking the absence of a signal at a position.

This section discussed various considerations for training and testing networks for a benchmark pattern recognition problem. The next section discusses the experiments and their corresponding results.

7.5 EXPERIMENTAL RESULTS

7.5.1 Overview

The theory of neural networks is not yet sufficiently matured to allow a network application to be precisely designed and developed. The process involves trial-and-error as well as research and planning. It starts with determining the most appropriate network to implement.

It is then followed by different aspects of network design such as deciding upon the network architecture and the learning algorithm parameters.

In the benchmark, the precise values for the network architecture and the learning algorithm parameters were all determined experimentally. Using demonstrative samples, this section discusses experimental results and some of the main points in the process of deciding upon a suitable configuration for the application.

7.5.2 The most appropriate neural network

Presently there are no hard rules available for choosing the right network for a given application. One way of finding the most appropriate network is to decide upon the training procedure required for the application. In the benchmark, because the exact output for every input in the training set was defined, supervised training was decided to be the most suitable. Hence networks such as the multi-layer feed-forward network with back-propagation (also referred to as the back-propagation network) and the Boltzmann machine (and its refinements) were likely candidates capable of mapping and producing new outputs. Counter propagation, was also a likely candidate because in training, the network receives the input and the target vector. The network of choice, as it is for most pattern recognition applications, was eventually decided to be a multi-layered fully connected feed-forward network with back-propagation (commonly known as a back-propagation network). As with all the practical work carried out, a large number of experiments, displaying the various aspects of the process were saved on hard disk for any further analysis.

7.5.3 Implementing back-propagation

- **Network architecture**

Simple two layered (i.e. input and output) networks, can represent the mapping with all representation already present in the input data because of the direct mapping of inputs to outputs. But they fail to learn the mapping if the data's innate representation is non-linearly separable such as the case for 'exclusive OR' function.

By adding intermediate layers, usually referred to as hidden layers, learning such arbitrary mappings is possible. The intermediate layer allows the network to develop its own internal

representation of the input patterns. The internal representation and the learning of features not explicitly obvious in the input data is constituted by the activations of the intermediate layers processing elements. They provide the network with an internal representation capability allowing it to decide upon whatever is important in representing the mapping. This is in contrast to relying on intrinsic relationships already built into the training data in two layer networks.

The first set of experiments, (see, Table 7.1), were concerned with seeking to apply back-propagation techniques to the experimental data. They involved many important changes within the network to find the most suitable configuration. These included varying the number of layers, the number of processing elements in the intermediate layer(s), and the learning parameters. Several networks were developed and then trained on data obtained from the rig. The generalised delta rule was used for the networks to learn to develop the required features to perform the desired mappings. The objective of the experiments was to find the most successful configuration to reach a minimum value for the average RMS error in the shortest number of training cycles. Testing results are compared and analysed in the performance section.

Sample experimental results demonstrating the various aspects of the process are included in Table 7.1. Except network number 6, which has two middle layers, they all have only one middle layer containing six processing elements. Deciding upon this configuration was achieved experimentally by varying the number of layers and the number of processing elements in the intermediate layer(s), while keeping all the other elements constant. All the networks were trained on a training file containing 63 patterns collected from the rig. To help the networks in their pattern recognition task, the 63 input patterns consisted of similar versions of 12 input training patterns.

At the start of training, weights for all of the 8 networks were initialised to random values between +1.0 and -1.0. Although with such a small range learning takes longer, this was thought necessary as the goal was to keep the activations close to 0 at the start of training. With larger values it was found that learning was generally shorter but the average RMS. error as well as the maximum output unit error tended to be higher.

Training was halted for networks 1, 2 and 3, after 120,000 cycles, as they were clearly stuck in a local minima. Training for the rest of the networks were terminated when their total RMS. error reached less than 0.05.

- **Varying the parameters**

  - Noise

Noise in the form of fluctuations that do not carry information is present in real world applications. It was thought that superimposing noise on data sets from the rig would accustom the networks to the environmental noise. Most networks in the experiments were trained using a significant amount of noise (up to 0.2, which makes it even difficult for a person to recognise the patterns).

The noise used in this set of experiments was calculated as a pseudo random number from a flat distribution with a maximum amplitude equal to the number entered. This meant that, say for a noise value of 0.1, each element of each input vector had a random value between -/+ 0.1 added to it independently. In the experiments, a multiplicative decay rate (e.g. 0.0001) was set for the noise to reduce gradually.

BP. NETWORKS [saved as] SPECI-FICATIONS	1 b1_63_ns	2 b2_63_ns	3 b3_63_ns	4 b4_63_ns	5 b5_63_ns	6 b6_63_ns	7 b7_63_ns	8 b8_63_ns
TRAINING CRITERIA	Training halted after 120000	Trainin halted after 120000	Trainin halted after 120000	Training_ stopped when max.out-put unit error <1	halted when max. output unit error <1	Training stopped when max.out-put unit error <1	Training stopped when max.out-put unit error <1	Training stopped when max.out-put unit error <1
NUMBER OF HIDDEN LAYERS	1	1	1	1	2 with similar specif.	1	1	1
LEARNING RATE [input to next layer]	0.9	0.9	0.7	0.1	0.5	0.5	0.3	0.1
LEARNING RATE [hidden to output layer]	0.9	0.9	0.7	0.1	0.5	0.5	0.7	0.1
MOMENTUM TERM [input to next layer]	0.9	0.5	0.7	0.1	0.5	0.5	0.2	0.9
MOMENTUM TERM [hidden to output layer]	0.9	0.9_	0.7	0.1	0.5	0.5	0.7	0.9
NOISE [+/-]	0.1	0.1	0.1	0.1	0.1	0.1	0.2	0.1
DECAY	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.001	0.0001
TOTAL RMS ERROR	0.37	0.25	0.28	0.04	0.03	0.04	0.02	0.02
MAXIMUM OUTPUT ERROR	0.99	0.99	0.99	0.16	0.09	0.09	0.09	0.09
CYCLES TRAINED	120000	120000	120000	1643	374	257	397	176
TESTING[no. of patterns correctly recognised on a testing file containing 18 patterns]	No testing	No - testing	No testing	12/18	14/18	12/18	13/18	14/18

Table 7.1 Results applying backpropagation

- Learning rate

True gradient descent in the total RMS error requires that, changes are made to the weights only in infinitesimal steps and only after the entire training set has been processed. (In order to use less RAM, some implementations of the BP algorithm make weight changes after each pattern). The proportion of the dictated weight changes may be set as a learning rate. The job of the learning rate constant is to set the speed of convergence of the weight vector to the ideal value at the bottom of the surface, or a more complex error function in multidimensional space. The larger the learning rate, the larger the weight changes and the faster the learning. The idea is to set the learning rate as high as possible, without causing the RMS. error to oscillate significantly. Generally, the optimum value of learning rate depends on the shape of the error function in weight space.

Mathematically, the learning rate constant must be less than 2.0, or the network cannot be stabilised, (Caudill and Butler, 1992). The learning constant must also be positive. If negative, the direction of the delta vector will be away from the ideal weight vector.

In practice, it was found that it was best to reduce the value of learning rate until the total RMS error showed a general decreasing value with time. Once it was close to the ideal position on the surface, even small differences between similar training samples when combined with a large learning constant, could cause the weight vector to move excessively and be slow to find the correct point.

- The momentum term

  One way to increase the learning rate without making the error oscillate is to add a momentum term to the weight change equation. Training proceeds much faster with this term. The term is a variable that determines the proportion of the last weight change that is added into the new weight change. It has the effect of preventing thrashing due to ripples in the error surface.

  Experimentally, (as it can be seen from network number 4), it was found that, a small value for the momentum term (e.g. 0.1) together with a small learning rate causes the network to converge very slowly. It was eventually concluded that a momentum value a lot higher (e.g. 0.9) together with a learning rate value (e.g. 0.1), so that they add up to 1.0, provide the best effect to the weight changes (see network number 8).

7.5.4 Further experiments to consider the more extreme cases

All the networks in Table 7.1 were trained on a training file containing 63 patterns which were normalised and scaled between -/+ 0.5. This was considered necessary as networks trained on raw data tended to end up in a local minimum. The networks in Table 9.1 were also assisted in their pattern recognition task by using input training patterns which consisted of similar versions of only a limited number (12) of input training patterns. As a result, one of the networks (number 8, Table 7.1) managed to learn in only 176 training cycles.

To consider the more extreme cases further experiments were undertaken. A main condition for a network to perform accurately is the quality and representativeness of the training cases. For training the networks in the following Table 9.2 a number of data sets were gathered. They consisted of 55 patterns for networks 9 and 10, 110 patterns for 11, 12 & 13 and 120 patterns representing the breadth of the problem, covering unusual and boundary cases for networks 14, 15 and 16. Network performance and test results presented in Table 7.2 are discussed in the performance section.

BP. NETWORKS [saved as] SPECI-FICATIONS	9 a5_55	10 a5_55_b	11 a5_110b	12 a8_110b	13 a5a_110b	14 a8_120b	15 a5_120b	16 a5a_120b
NUMBER OF HIDDEN LAYERS	2 with similar specifications.	2 with similar specifications.	2 with similar specifications.	1 hidden layer only	2 with similar specifications.	1 hidden layer only	Two with similar specifications.	2 with similar specifications.
TRAINING DATA	Raw training data consisting of 55 patterns.	55 patterns scaled [-/+ 0..5];	110 patterns normalised & scaled[-/+ 0.5].	110 patterns normalised & scaled[-/+ 0.5].	110 patterns normalised & scaled[-/+ 0.5].	120 patterns normalised & scaled[-/+ 0.5].	120 patterns normalised & scaled[-/+ 0.5].	120 patterns normalised & scaled [-/+0.5]
LEARNING RATE [input to next layer]	0.5	0.5	0.5	0.1	0.1	0.1	0.5	0.1
LEARNING RATE [hidden to output layer]	0.5	0.5	0.5	0.1	0.1	0.1	0.5	0.1
MOMENTUM TERM [input to next layer]	0.5	0.5	0.5	0.9	0.9	0.9	0.5	0.9
MOMENTUM TERM [hidden to output layer]	0.5	0.5	0.5	0.9	0.9	0.9	0.5	0.9
TOTAL R.M.S. ERROR	0.4	0.04	0.04	0.04	0.05	0.02	0.03	0.03
MAXIMUM OUTPUT ERROR	0.69	0.09	0.09	0.09	0.1	0.09	0.09	0.09
CYCLES TRAINED	120000	708	545	1139	358	1192	960	618
TESTING [no. of patterns correctly recognised on a testing file containing 18 patterns]	Training halted after 120000 cycles; no testing	12/18	15/18	14/18	15/18	15/18	18/18	18/18

Table 7.2: Applying back-propagation to data covering more extreme cases

7.6 ANALYSIS OF DEVELOPED MODELS ON TEST DATA

7.6.1 Training patterns

Neural networks learn relatively better when their inputs are simplified. This is demonstrated by networks in Table 7.1. Training files for all these networks were normalised and scaled between -/+ 0.5. Networks such as number 9 in Table 7.2 which were trained on data not normalised, mostly failed to train. Others (e.g. network number 10) trained on data which were only scaled within a certain range, still had difficulty in learning.

Networks 1 to 8, in Table 7.1 were additionally assisted in their 'pattern recognition' learning task by using input training patterns which consisted of similar versions of only a dozen input training patterns. As a result, one of the networks (number 8) managed to learn in only 176 training cycles. However, when tested on unseen data they all performed poorly. For example, in the case of network number 8, it failed to recognise 4 out of 18 test patterns. This is because an adequate number of training patterns was not available to allow the networks to cover the problem domain.

To find the least number of inputs to adequately represent the problem, some networks were trained on sequentially larger numbers of different input patterns. As can be seen from the results in Table 7.2, training took longer but performance on testing was improved. Networks 14, 15 and 16 which were trained on 120 different patterns representing the breadth of the problem, performed relatively well. Networks 15 and 16 managed to correctly recognise all of the test patterns, from a test file containing 18 unseen patterns.

7.6.2 Architecture

Choosing the right number of hidden processing elements and layers for the application was carried out through a process of trial-and-error. The process was time consuming, tedious and at times contradictory. For example, increasing the number of hidden processing elements generally resulted in longer learning. Sometimes, however, increasing the numbers slightly actually resulted in decreasing learning time. But generally, too many processing elements resulted in poor generalisation. That is the networks tended to memorise the training set rather than learn it. Defining too few hidden processing elements mostly resulted in incorrect classification. Experimentation eventually lead to decide upon 6 hidden processing elements for networks with one hidden layer presented in Tables 1 and 2. All the

networks with two hidden layers (e.g. 15 and 16) have 6 processing elements in the first hidden layer and 4 in the second.

Only a few networks with more than two hidden layers were constructed. It has been proved by many researchers (e.g. Blum and Li, 1991) that a continuous function can be well approximated by a static neural network with a single hidden layer, where each processing element in the middle layer has a continuous sigmoidal non-linearity. That is, there is no need to build networks with very many middle layers. Despite such results, further work needs to be done to investigate the relationship between accuracy of the function being approximated with the number of hidden layers. Chester (1990) has pointed out that neural networks with two hidden layers appear to provide higher accuracy and better generalisation than a network with a single hidden layer. For the benchmark, this has actually been confirmed. That is networks with two hidden layers (e.g. 15 and 16 in Table 7.2) performed much better than networks with a single hidden layer.

To accustom such networks, to the noise typically present in a process control environment, a random noise value between -/+ 0.1 was added to each element of each of their input vectors independently. Network 16, managed to train in only 618 cycles. For testing, a file containing 18 unseen new patterns was used. Both networks 15 and 16 managed to correctly recognise all the test patterns.

7.7 OTHER IMPLEMENTATIONS

7.7.1 Implementing and testing recurrent back-propagation

Networks developed in the preceding section are static, feed-forward, or non-recurrent networks. They are able to approximate non-linear functions to a desired degree of accuracy. As a result they have been used to model dynamic systems. However, the use of dynamic, feedback, or recurrent networks to represent dynamic systems may be more appropriate. A recurrent network may be able to do a lot more than a back-propagation network with the same number of weights (Hecht-Nielsen 1987). No theorem confirming this yet exists. The feedback in recurrent networks implies local memory characteristics (Siac, 1989). But, it may cause instability. In a stable network, successive iterations produce smaller and smaller output changes until eventually the outputs become constant. In an unstable network, the process may never end. Chaotic systems, which have interesting properties of their own, are one example of an unstable network.

In the recurrent back-propagation architecture used for the application, in addition to the feed-forward connections from the input layer to the hidden layer and from hidden layer to the output layer, feedback connections were made between hidden units and between output units. That is, all the processing elements in these layers have feedback connections to themselves and to every other processing element in that layer. As a result, the generated outputs depend not only on the current inputs but also on the current state of the units in layers with feedback connections. The feedback loops in the output layer are really there to allow the network to learn relationships between successive output vectors in a sequence (as potentially such a relationship can exist in the benchmark). More importantly, however, are the feedback loops in the hidden layer which allow the network to form a "memory trace" of the input sequence. Such architectures with feedback are ideally suited for control and modelling (identification) of the forward or inverse dynamics of a chemical process control system.. This is so because, such real world systems requiring to be modelled, are themselves non-linear dynamical systems.

The experiments in Table 7.3 are concerned with seeking to apply the recurrent back-propagation network to the experimental data within a noisy environment. Networks 17 and 18 were trained on data consisting of 110 patterns representing the problem. As can be seen from the results, network no.17 did not do well on testing. Network 18, with a more suitable configuration for the application, performed slightly better. Networks 19, 20 and 22 which were trained on a data file containing 120 patterns, representing the breath of the problem, performed much better.

BP. RECURRENT NETWORKS [saved as] / SPECI-FICATIONS	17 A_BPR1	18 A_BPR2	19 A_BPR3	20 A_BPR4	21 A_BPR5 This net is the same as net 20 but training halted earlier	22 A_BPR6	23 A_BPR7 This net is the same as net 22 but training halted earlier
NOISE [+/-] / DECAY	0.01 0.001	0.01 0.001	0.01 0.001	0.1 0.001	0.1 0.001	0.1 0.002	0.1 0.002
TRAINING DATA	110 patterns normal-ised & scaled [-/+ 0.5].	110 patterns normalised & scaled [-/+ 0.5].	120 patterns normalised & scaled [-/+ 0.5]	120 patterns normalised & scaled [-/+ 0.5]	120 patterns normal-ised & scaled [-/+ 0.5]	120 patterns normal-ised & scaled [-/+ 0.5]	120 patterns normalised & scaled [-/+ 0.5]
LEARNING RATE CONS-TANT [input to next layer]	0.2	0.1	0.1	0.1	0.1	0.1	0.1
LEARNING RATE CONS-TANT [hidden to out putlayer]	0.2	0.1	0.1	0.1	0.1	0.1	0.1
MOMENTUM TERM CONS-TANT [input to next layer]	0.8	0.9	0.9	0.9	0.9	0.9	0.9
MOMENTUM TERM CONS-TANT [hidden to output layer]	0.8	0.9	0.9	0.9	0.9	0.9	0.9
TOTAL R.M.S. ERROR	0.05	0.04	0.04	0.03	0.04	0.03	0.04
MAXIMUM OUTPUT UNIT ERROR	0.1	0.09	0.1	0.09	0.19	0.09	0.19
CYCLES TRAINED	1231	1381	1962	1988	461	832	205
TESTING [no. of patterns recognised correctly from a testing file containing 18 patterns]	13/18	14/18	16/18	18/18	18/18	18/18	18/18

Table 7.3: Applying recurrent backpropagation to the experimental data

Using a testing file containing 18 unseen patterns, network 19 managed to correctly recognise 16 patterns. Networks 20 and 22 have the same configuration as 19. They were trained on the same training file as 19, but in a much noisier environment (started at 0.1 rather than 0.01). As a result they managed to correctly recognise all the test patterns. Network 20 managed the training in 1988 cycles. To improve upon this noise was reduced in bigger chunks for network 22. That is, a decay rate of 0.02 rather than 0.01 was used. As a result the number of training cycles for network 22 was reduced to 832. The stopping criteria for training the networks 20 and 22 was for the value of 'the maximum output unit error' to reach

0.1 or less. By increasing this value to 0.2, these networks managed to recognise the patterns satisfactorily, after training for a much reduced number of cycles. These results are presented in 21 and 23 respectively. As can be seen from the results in Table 7.3, network 23 correctly recognised all the unseen test patterns after training for only 205 cycles.

By applying noise randomly to each input pattern just before it was presented to the network (each time), the network only saw a series of approximations of that input. Such a technique forces the network to generalise, one of the key goals. As Table 7.3 demonstrates, using slightly noisy data effectively improved the training time and the learning.

The above results demonstrate the importance of noise in training. This is in sharp contrast to conventional systems, and demonstrates the neural network's potential in dealing with the noisy data that occurs in process control applications.

7.7.2 Implementing Boltzmann machine and counter-propagation

- **Boltzmann machine**

The architecture of the Boltzmann machine used was the same as the back-propagation networks used earlier, except that the processing elements had bistate values (0 or 1). The hidden units, like those of a multi-layer back-propagation network, act as feature detectors. They learn using an energy state optimisation method called simulated annealing. This approach to learning allows them to perform optimisation as well as pattern recognition tasks.

The version of the Boltzmann machine used was a modification by Hinton and Sejnowski (1986). Unlike the original Boltzmann Machine, where the weights between units are calculated directly from the set of patterns, the weights in this version were slowly modified by a statistical process. Using data gathered from the Bytronic rig. several networks were trained. Their operation after training was similar to the back-propagation networks developed earlier. However, convergence generally took orders of magnitude (at least two) more than the back-propagation network.

- **Counter-propagation**

The Counter Propagation Network (CPN), developed by Hecht-Nielsen (1987), combines Kohonen learning and Grossberg learning. The resulting network functions as a statistically optimal self-programming lookup table.

For the benchmark problem, CPN did not perform as well as backpropagation, but converged more rapidly. Figure 7.1 presents a sample counter propagation network trained on a data file containing 120 patterns collected from the rig. It consists of five layers: *[1]* the input vector 'x', *[2]* the associated target vector 'y', *[3]* the Kohonen middle layer, *[4]* and *[5]* the output vectors from the network. During training *[1]* and *[2]* receive the input and target vectors. Using the connection weights *[1]* to *[3]* and *[2]* to *[3]* (see *'Kohonen 1 to 3'* and *'Kohonen 2 to 3'* in Fig. 7.1 respectively), these are connected to *[3]* the middle (Kohonen) which maps these input vectors into a topological space where nearest neighbour relationships are preserved (see Kohonen, 1984 for detail). Once the topology preserving map is formed, the winning unit is passed through sets of Grossberg weights to form the network output *[4]* and *[5]*. Grossberg weights employ the outstar learning paradigm developed by Grossberg (1982). As with the backpropagation tests, the introduction of noise actually assisted the network during training. After training, the network performed well but not as good as that of BP networks presented earlier. However, it was able to recall patterns in both directions.

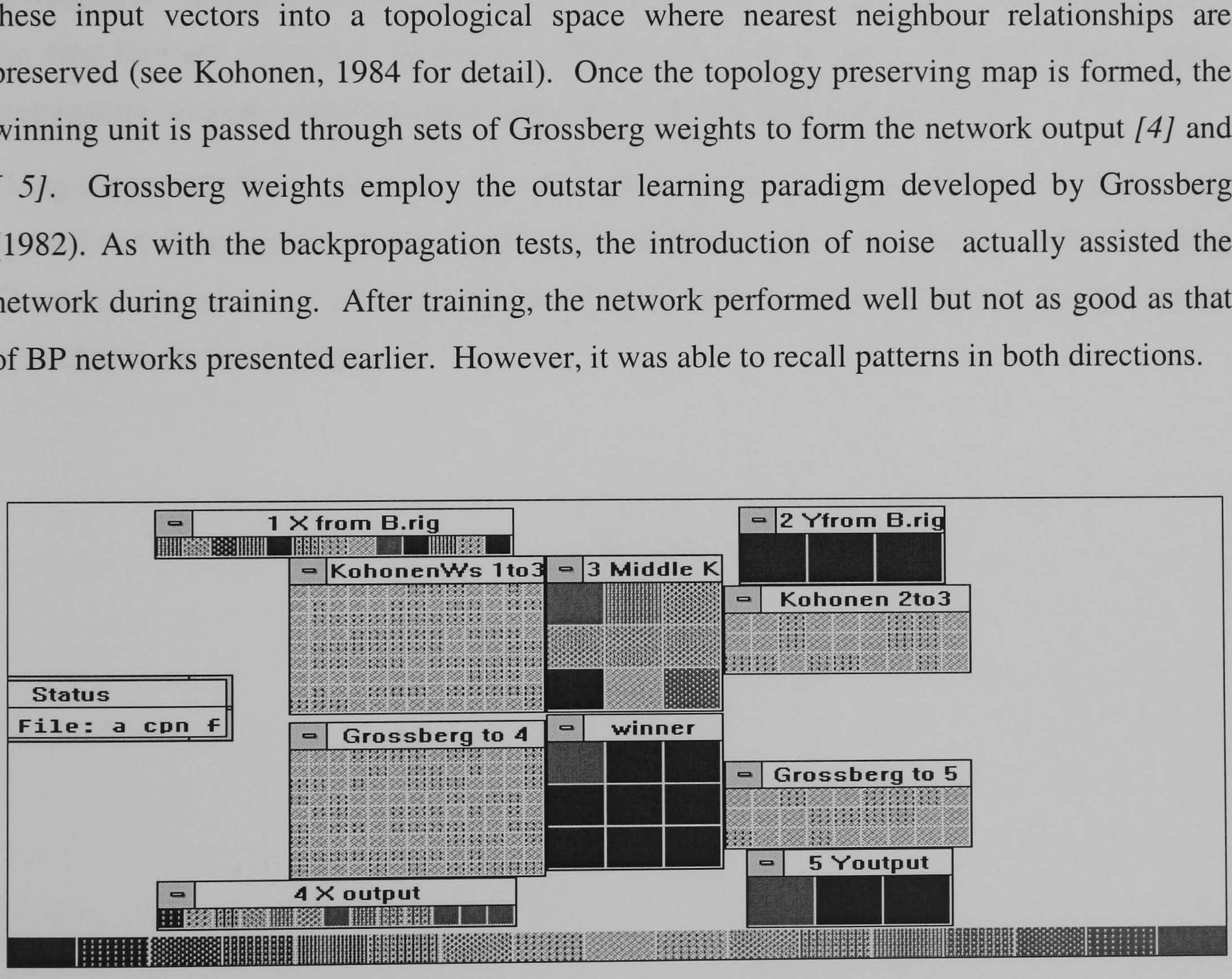


Figure 7.1: Applying a counter propagation network to the training data

7.8 SUMMARY

Experimental results and some of the main considerations in the process of deciding upon a suitable configuration for the benchmark, have now been discussed. Applying a small amount of noise randomly to each input pattern actually forced the networks to generalise more readily. In process control applications such robustness to noisy data is a main issue.

The network of choice for our pattern recognition application was considered to be a multi-layered fully connected feed-forward network trained with back-propagation. Networks with two intermediate layers particularly performed well.

However, to represent dynamic systems, it was suggested that using feed-forward networks extended with feedback connections may be more appropriate. Such dynamic models were developed by adding feedback connections between hidden units and between output units of the feed-forward networks. Dynamic networks are able to do a lot more than a back-propagation network with the same number of weights (Hecht-Nielsen 1987).

Using the knowledge on the networks, which performed well, artificial neural networks were applied to a digital communication application, details of which are included in the following Chapters. Comments on all the experimental work carried, as well as general conclusions are presented in the final Chapter.

8 APPLICATION TO DIGITAL SIGNAL COMMUNICATION

8.1 OVERVIEW

Using a pattern recognition problem, the previous Chapter addressed issues regarding noise, generalisation, and incorporation of approximate neuro-models. It was found that careful use of noisy data, may effectively improve model performance. The experimental results in Chapter 9 are used in developing a neuro interface engine for a digital signal communication system. This Chapter, introduces the digital signal communication system considered. The system employs fractal modulation to code data into a format that can securely be transmitted (Blackledge et al., 1996). Integrating neuro-filters with the digital communication system enables the system to operate consistently at lower signal/noise ratios, making it commercially viable. Following an introduction to the system, noise in communication systems, noise modelling and computing, and details of computing fractal noise, utilised to produce neuro-filters, are presented.

8.2 DIGITAL COMMUNICATION EMPLOYING FRACTAL MODULATION

Digital communication using fractal modulation rather than conventional modulation techniques, such as amplitude, frequency or phase, is a hot topic of research. Using fractal modulation, a technique of coding binary data in a form indistinguishable from the background noise, for digital transmission and reception of sensitive information is considered (Blackledge et al., 1996). In their work, random scaling fractal signals were used to model background noise in which the main characteristic of fractal signals, the fractal dimension, was used to describe the texture of the fractal. Figure 8.1(a) shows an example of a fractal coded binary signal with 3 fractals per bit, each with 64 samples with a fractal dimension of 1.1 for bit 0 and 1.9 for bit 1. In the case shown in Figure 8.1 (b), the change in signal texture between 0 and 1 is reduced by reducing the difference in fractal dimension.

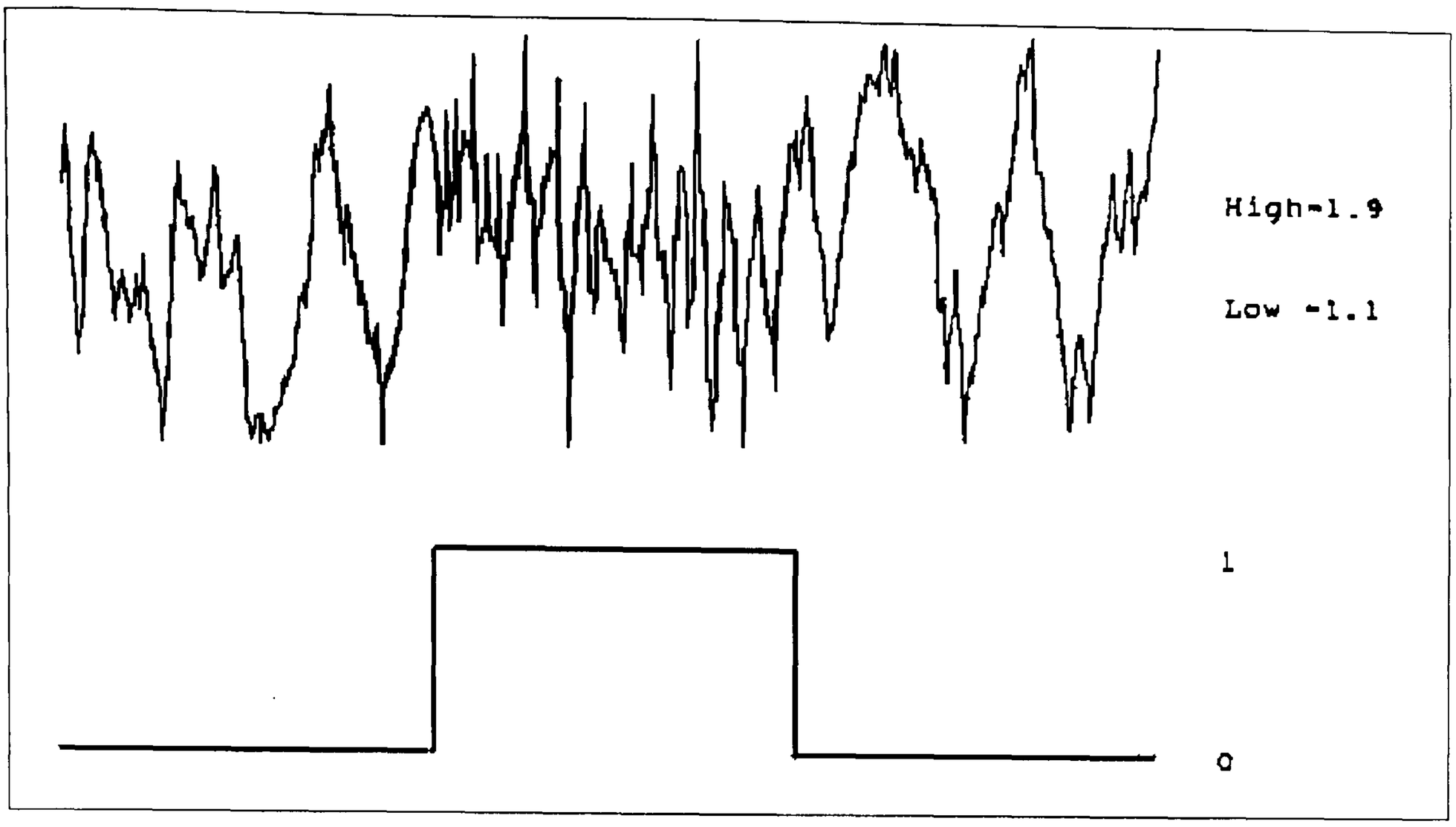


Figure 8.1 (a)
A fractal coded binary signal with 3 fractals per bit, with
a fractal dimension of 1.1 for bit 0 and 1.9 for bit 1.

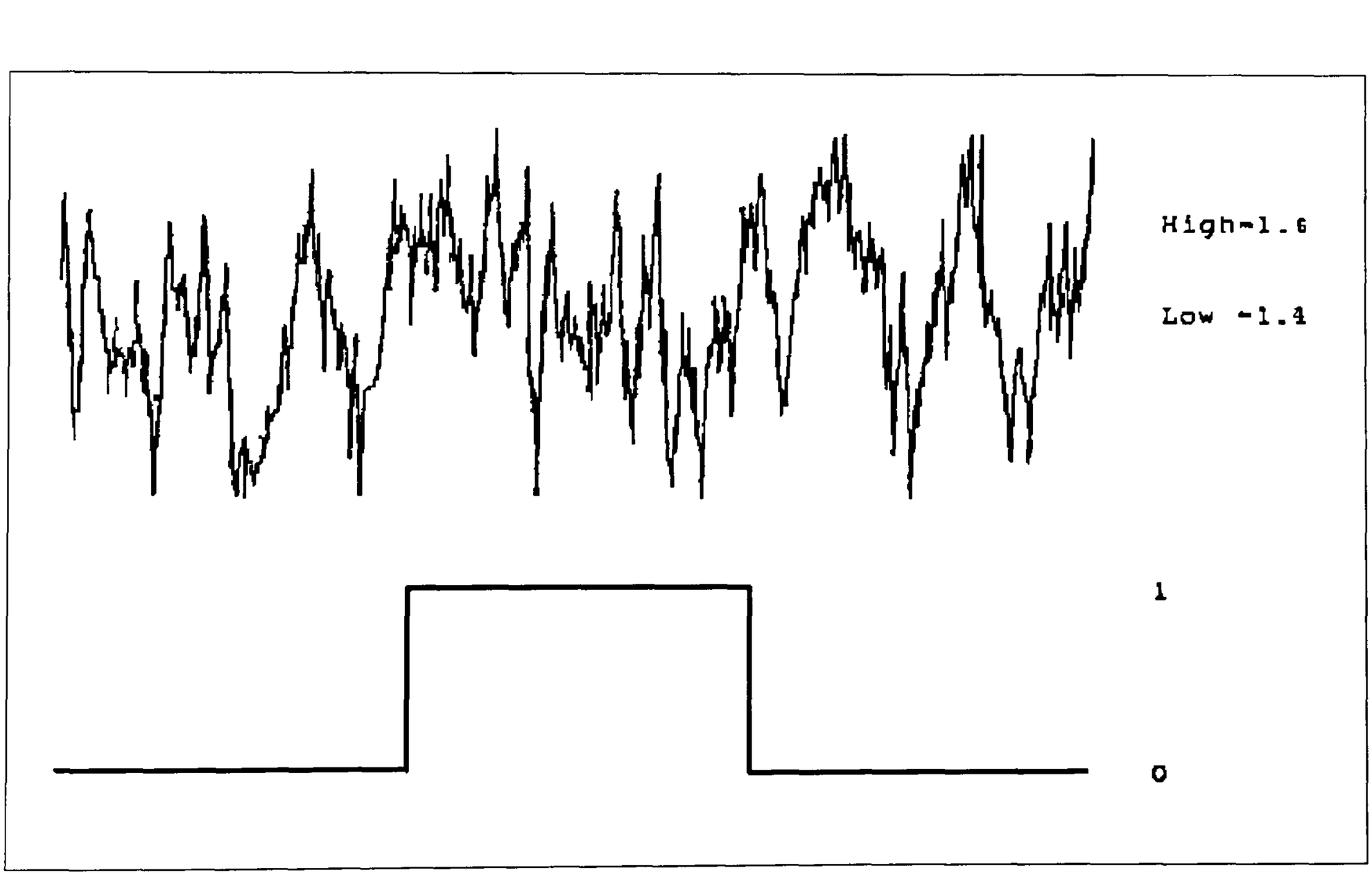


Figure 8.1 (b)
A fractal coded binary signal with 3 fractals per bit, with
a fractal dimension of 1.4 for bit 0 and 1.6 for bit 1.

8.2.1 System performance

The performance of any system based on transmitting and receiving bit streams, or binary sequences, is ultimately effected by transmission noise. That is, the information is transmitted in binary form, but on reception the binary sequence is convolved with transmission noise. Additionally, the performance of the system is subjected to changes in the fractal generating parameters.

- **Examples of output**

Examples of the output produced by the system, using 64 samples per fractal, are shown in Figure 8.2 (a) and Figure 8.2 (b). The examples show the fractal signal and the reconstructed fractal dimension superimposed on the original code, top dotted line. The original and estimated binary sequences are displayed on the right hand side. In the first example, Figure 8.2 (a), with 1 fractal per bit, and lower and upper fractal dimensions of 1.10 and 1.90, two errors have occurred out of a total of 100 bits. In the second example, Figure 8.2 (b), with 5 fractals per bit, and lower and upper fractal dimensions of 1.60 and 1.90, there is only one error .

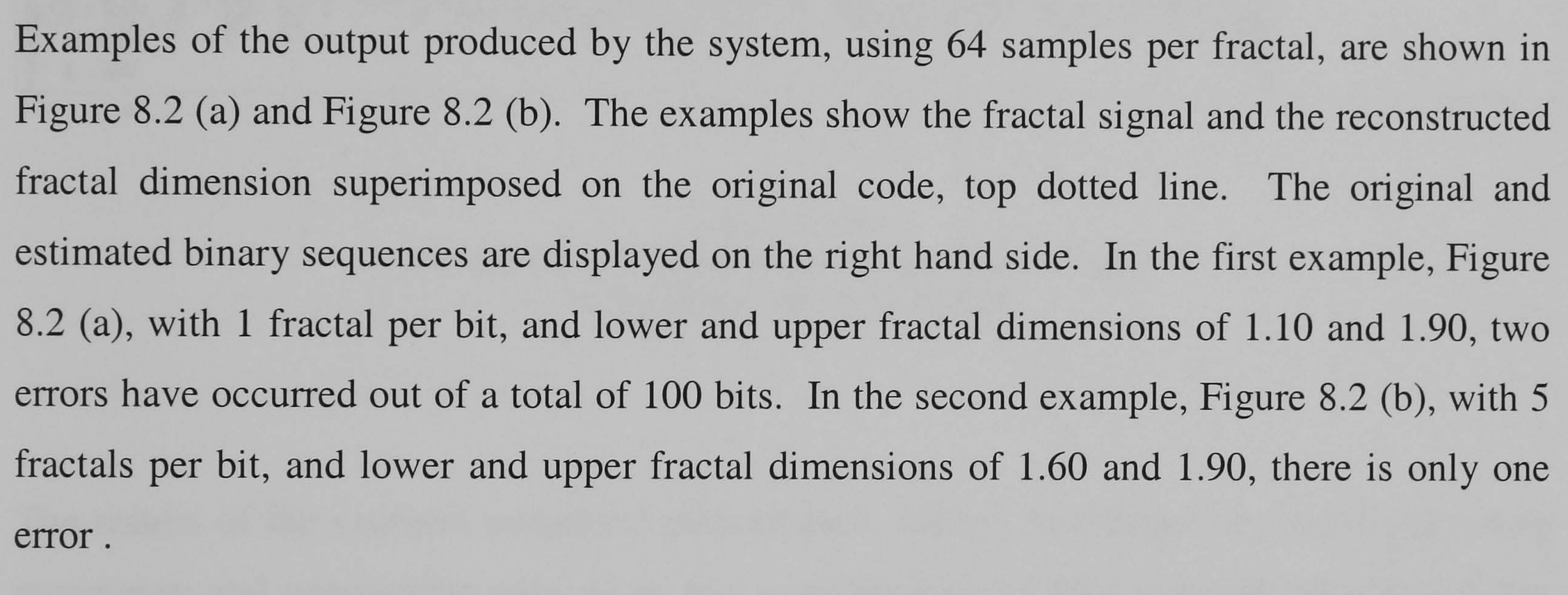


Figure 8.2 (a)
An example with 1 fractal per bit, and lower and upper fractal dimensions of 1.10 and 1.90

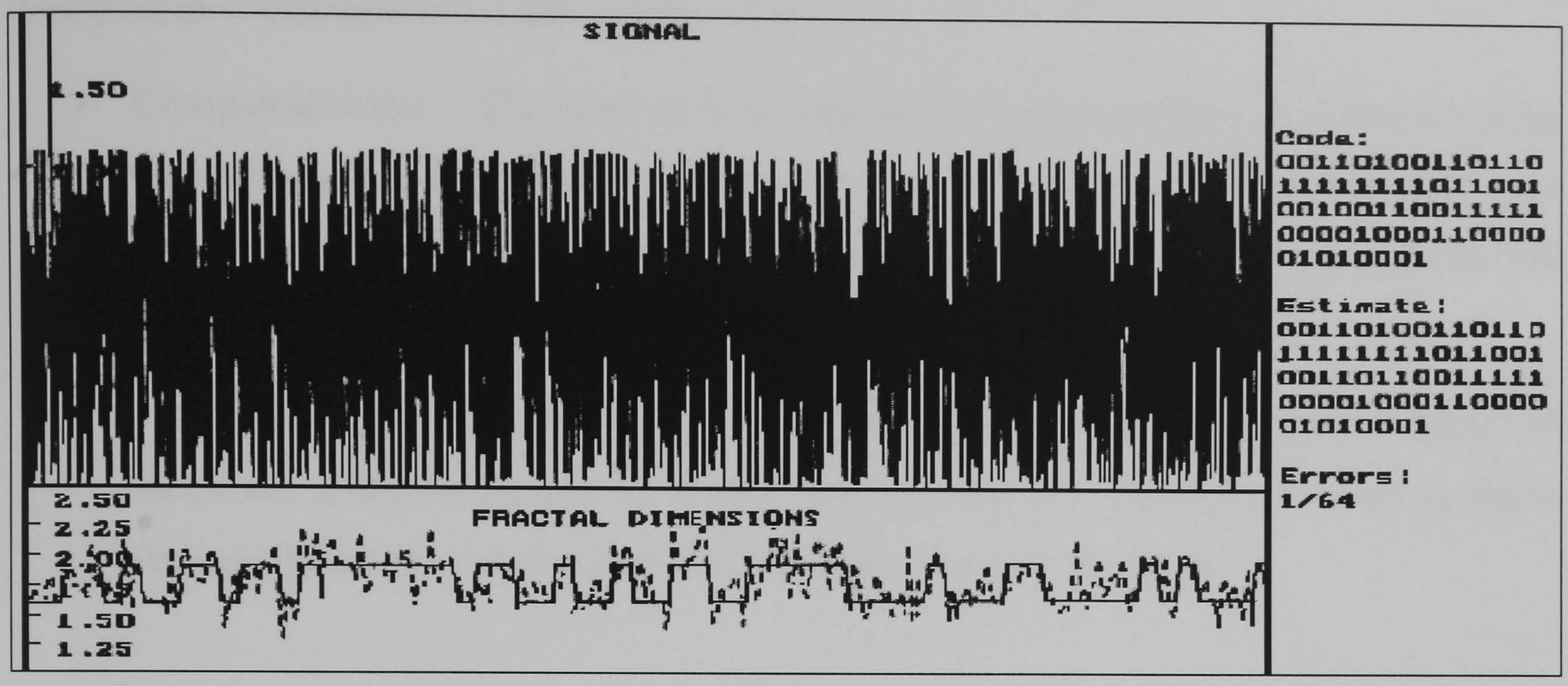


Figure 8.2 (b)
An example with 5 fractals per bit, and lower and upper
fractal dimensions of 1.6 and 1.9

The results of the system's numerical performance, subject to changes in fractal generating parameters and signal/noise ratio, show that a combination of wide intervals between the two fractal dimensions with a large number of fractals per bit achieve greater accuracy (Blackledge et al., 1996).

8.3 NOISE

8.3.1 Digital communication

In any system in which the transmission of data is the main consideration, noise is a major issue. Transmission noise affects both digital and analogue systems and designers are always trying to find new methods to reduce it. There are two classes of transmission noise, internal and external.

- **Internal noise**

Internal noise affects the signal as it propagates through the system and circuits. This noise cannot be eliminated but can be controlled through circuit design. Internal noise can be classified as component noise and system noise.

- Component noise - The types of noise that cause component noise, e.g. thermal noise caused by agitation of electrons, or noise caused by fluctuations in the conductivity of the semiconductor material, have the same amplitude for all frequencies across the whole frequency spectrum, i.e. white noise.
- System or equipment noise - This is caused by the non-linear characteristics of electronic components and by circuits such as filters. This type of noise produces frequencies in the output signal that were not present at the input.

- **External noise**

Once a signal is transmitted, it comes under the influences of external sources of noise that cannot be controlled, e.g. radiation, or static interference such as volcanic activity, dust storms and sand storms. Instead, this should be taken into account when designing the receiver.

8.3.2 1/*f* phenomena in physical and biological processes

The inverse relationship between frequency (*f*) and squared amplitude or power in the Fourier spectrum, '$1/f^{\alpha}$ power spectra' where the exponent 'α' is a value between 0.5 and 1.5, is one of the most common fractals found in nature (Voss, 1979).

In electronics systems 1/*f* noise is also referred to as flicker noise or pink noise. In such systems, this intermittent bursting behaviour in a time series, is one of the most common forms of noise.

Mathematical properties:

- In general, 1/*f* noise is fractal with respect to time.
- 1/*f* spectra have the largest values of dependency or correlation with the past of any of the random scaling noises, e.g. white or brown. This dependency structure of a 1/*f*

spectrum, or memory, implying correlation over all past times is based on the processes of fractional integration of noise.

- Fractional integrals are additive, e.g. applying the ½ integral twice gives a full integral. In terms of spectral analysis, quantifying the ½ integral involves adding up all components from the past. For example, $1/f$ can be generated using half-order integration of Gaussian white noise. The power spectrum of this noise is a constant and the current value of a Gaussian white noise at one point in time is totally independent of a past value.
- Full order integration of a Gaussian white noise, or the summation of random steps, gives a Brownian motion, or $1/f^2$ noise in the power spectrum.
- Half order integration consists of linear convolution with a kernel, or Green's function
- $1/f$ processes have underlying Lεvy stable distributions which have the same additive properties as Gaussian distributions, i.e. the some of two $1/f$ processes gives a $1/f$ process of a higher order (Takayasu, 1990).
- The product of $1/f$ processes is also a $1/f$ process (Kawai, et al., 1993).

Using fractal noise, robust neuro models were developed to identify noisy outputs from the communication system described in this Chapter.

8.4 MODELLING AND COMPUTING NOISE

8.4.1 Models and characteristics

- **Models**

The ideal approach for developing a model is to:

(a) analyse physics of the system,

(b) establish appropriate set of differential equations,

(c) solve equations,

(d) compare solution with experimental results, and

(e) back to (a) to refine model.

But the physical origins of many noise types are not well understood. Secondly, conventional approaches for modelling noise fields usually fail to accurately predict their characteristics.

- **Characteristics**

Defining the characteristics of noise, the principal approaches:

(a) Probability Distribution Function (PDF)- shape of distribution of amplitudes of noise sequence

(b) power spectrum of noise – shape of Power Spectral Density Function (PSDF)

Based on this, noise has two principal characteristics:

(a) power spectrum characterised by irrational power laws, and

(b) exhibits statistical self affinity.

8.4.2 Phenomenological approach

- **Examples of PSDFs for different stochastic processes**

These include:

(i) Bermann process

$$P(\omega) = \frac{A\omega^{2g}}{\left(\omega_0^2 + \omega^2\right)^q}$$

(ii) Ornstein-Uhlenbeck process

$$P(\omega) = \frac{A|\omega|}{\left(\omega_0^2 + \omega^2\right)}$$

(iii) Fractional Brownian Motion

$$P(\omega) = \frac{A}{\omega^{2q}}$$

- **Finding a PSDF which provides a general model for different types of noise.**

Following M.Sc. notes by Professor Blackledge, a PSDF of the following form is considered.

$$P(\omega) = \frac{A\omega^{2g}}{\left(\omega_0^2 + \omega^2\right)^q}$$

where

ω - angular frequency

ω_0 - *characteristic frequency,*

A - constant (*a scaling factor*),

$q>0$ *and* $g>0$

8.4.3 Computing noise

Complex spectrum of noise can be written as

$$N(\omega) = H_{gq}(\omega)\, W(\omega)$$

$$H_{gq}(\omega) = \frac{B(i\omega)^g}{\left(\omega_0 + i\omega\right)^q}$$

where

g and q are positive floating point numbers,

H_{gq} *is the transfer function given by* ($B = \sqrt{A}$),

A is a scaling factor,

$W(\omega)$ *is the complex spectrum of 'White Noise'* (*i.e. noise with a zero mean Gaussian distribution whose* PSDF *is a constant.*),

The noise field n(t),

as a function of time t,

is given by the inverse Fourier transform of $N(\omega)$

i.e.

$$n(t) = \tfrac{1}{2\pi}\int_{-\infty}^{\infty} N(\omega)\exp(i\omega t)\, d\omega$$

$$n(t) = \mathrm{Re}\ \hat{F}^{-1}\left[N(\omega)\right]$$

where $\hat{F}$ – *Fourier operator*

$\hat{F}^{-1}$ – *Inverse Fourier Transform*

For $\omega > 0$ and $q > g$, *the maximum value of* $P(\omega)$ *occurs when*

$$\omega = \omega_0 \sqrt{g/(q-g)}\ .$$

The value of $P(\omega)$ *at this point is*

$$P(\omega) = A\omega_0^{\,2(g-q)}\,\frac{g^g}{q^q}(q-g)^{q-g}$$

Beyond this point, the PSDF *decays and its asymptotic is dominated by a* ω^{-2q} *power law which is consistent with conventional random fractal signals. At low frequencies, the* PSDF *is characterised by the term* ω^{2g} .

8.4.4 Fractional differential equations

The noise $n(t)$ has been expressed in terms of the inverse Fourier transform of $N(\omega)$

Given

$$N(\omega) = \frac{B(i\omega)^g}{(\omega_0 + i\omega)^q}\ W(\omega)$$

Fourier inversion leads to (casual case)

$$n(t) = \frac{B}{\Gamma(q)} \int_{-\infty}^{t} \frac{e^{-\omega_0(t-\tau)}}{(t-\tau)^{1-q}} \frac{d^g}{d\tau^g}\omega(\tau)\, d\tau$$

where

$$\omega(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} W(\omega)\exp(i\omega t)\, d\omega$$

$$= \hat{F}^{-1}\left[W(\omega)\right]$$

This result is based on the following definition for a fractional differential of a function $f(t)$

$$\frac{d^g}{dt^g} f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega)(i\omega)^g \exp(i\omega t)\, d\omega$$

where

$F(\omega)$ *is the Fourier transform of* $f(t)$.

8.4.5 Random scaling fractal signals

Given the power spectral density function

$$P(\omega) = \frac{A\omega^{2g}}{\left(\omega_0^2 + \omega^2\right)^q}$$

a random scaling fractal signal is obtained when g = 0 and $\omega_0 = 0$.

It can then be written

$$P(\omega) = \frac{A}{\omega^{2q}}$$

where q is defined in terms of the Fractal Dimension D ($1 < D < 2$) (the Minkowski of Box Counting Dimension) by

$$q = \frac{5 - 2D}{2}$$

This result is consistent with the spectral noise model, ignoring scaling constant A.

$$N(\omega) = \frac{W(\omega)}{(i\omega)^q}$$

or

$$n(t) = \hat{R}[n(t)] = \frac{1}{\Gamma(q)} \int_{-\infty}^{t} \frac{\omega(\tau)}{(t-\tau)^{1-q}}\, d\tau$$

Riemann-Liouville (fractional) integral

- **Remarks**
  - $n(t)$ can be considered to be a solution to the fractional differential equation

$$\frac{d^q}{dt^q} n(t) = \omega(t); \frac{1}{2} < q < \frac{3}{2}$$

- The Riemann-Liouville (fractional) integral has the following fundamental property

$$n'(t) = \hat{R}[\omega(\lambda t)] = \frac{1}{\lambda^q} n(\lambda t)$$

or

$$Pr[n'(t)] = \frac{1}{\lambda^q} Pr[n(\lambda t)]$$

describes, statistical self affinity, and where

Pr [ ] - *denotes the PDF.*

8.5 FRACTAL NOISE FOR DEVELOPING NEURO-FILTERS

Using '$1/f^q$ power spectra' where the exponent 'q' is a value between 0.0 and 1.5, noise with the following description was produced (Table 8.1).

Order of integrodifferentiation (q)	Description
$q = 0$	$1/f^0$ - Gaussian white noise
$0 < q < 0.5$	Non-specific coloured noise
$q = 0.5$	$1/f$ fractional noise – Pink
$0.5 < q < 1$	Anti-persistent fractional Brown process
$q = 1$	Non-persistent Brown process
$1 < q < 1.5$	Persistent fractional Brown process

Table 8.1 Noise description

The generated noise was then used to develop neuro-filters to correctly identify noisy signals received by the digital communication system.

Algorithm for developing neuro-filters using fractal noise

Step 1. *Compute White Noise* ω_i ; $i = 1,2,\ldots, N$

Step 2. *Compute DFT of* ω_i *giving* W_i *using FFT algorithm.*

Step 3. *Filter* W_i *with* $1/\omega_i^q$;

Step 4.*Inverse DFT result using FFT giving* n_i .

Step 5. *Convolve fractal noise with training data*

Step 6. *Develop neuro paradigms using convolved training/testing data*

8.5.1 Examples

Figure 8.3 (a) shows examples of white (q = 0), pink (q = 0.5), and brown noise (q = 1) produced for developing robust neural networks, where q is the order of integro-differentiation with respect to white noise. Power spectrums of white, pink and brown noise are included in Figure 8.3 (b). Power spectrum is useful in verifying the type of noise produced. For example, a log-log plot of 'power spectrum verses frequency of white noise (Figure 8.4) gives a gradient of zero. A different gradient, e.g. –1, indicates a different noise, e.g. pink noise.

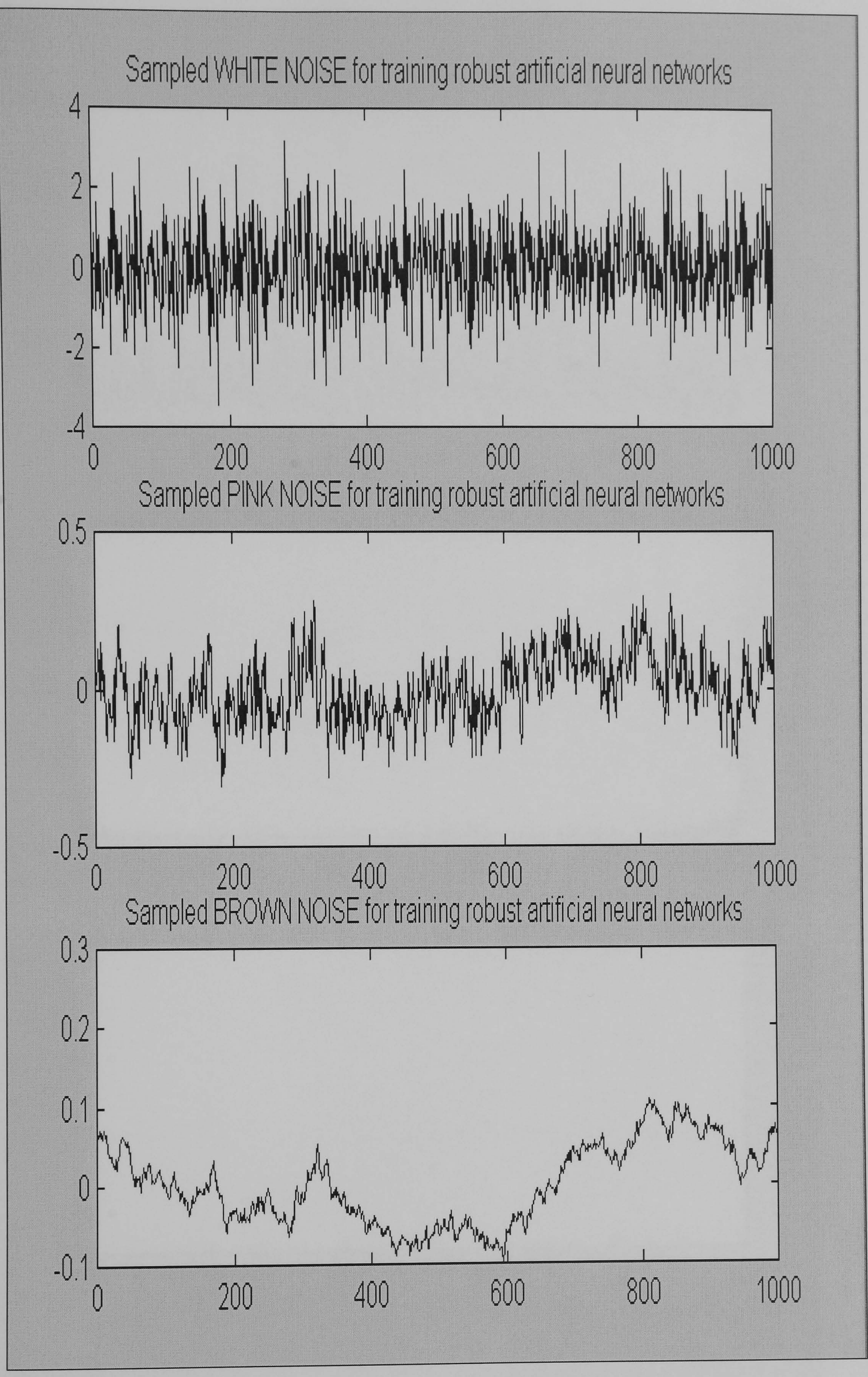


Figure 8.3 (a)
Examples of white, pink and brown noise produced

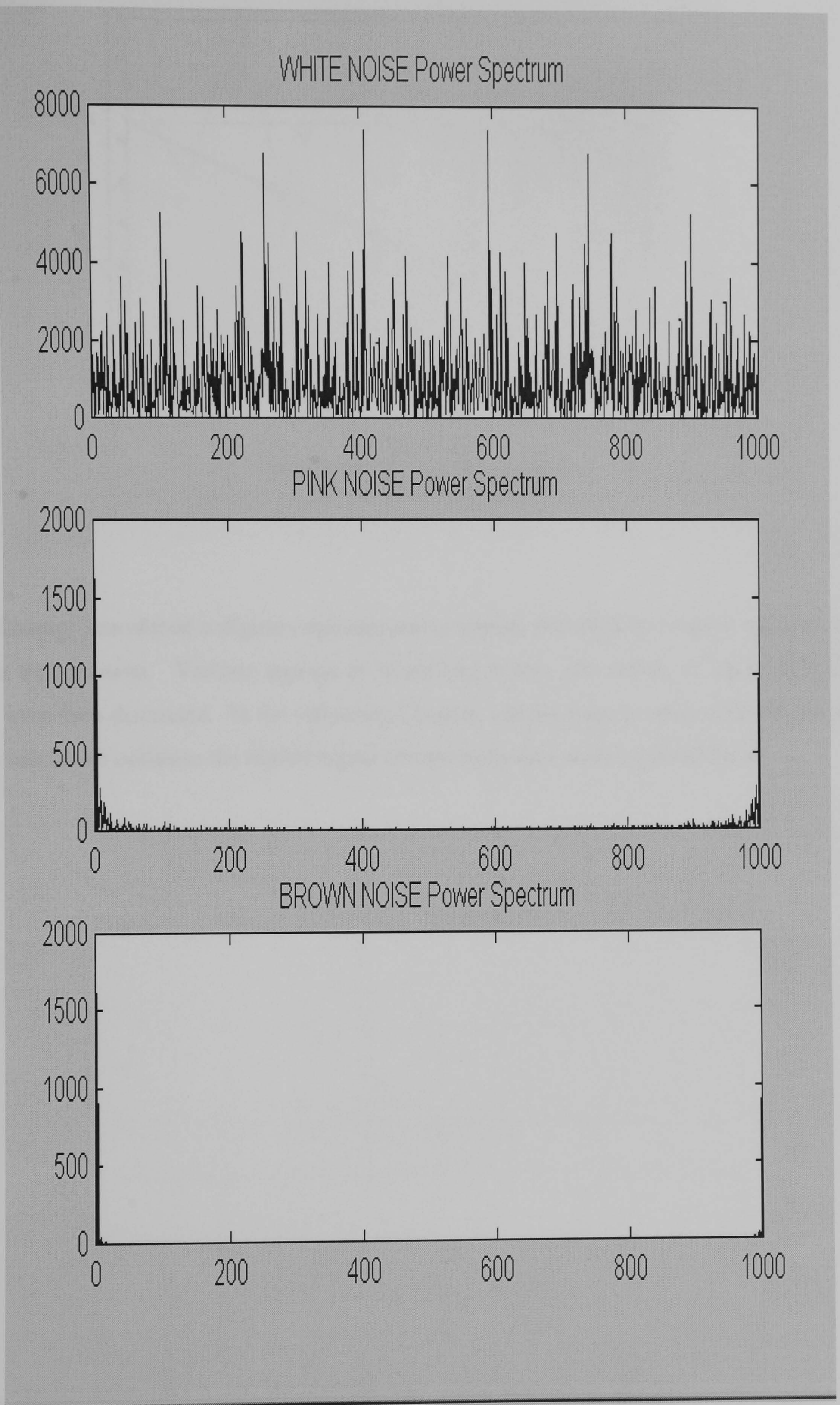


Figure 8.3 (b)
White, pink and brown power spectrum

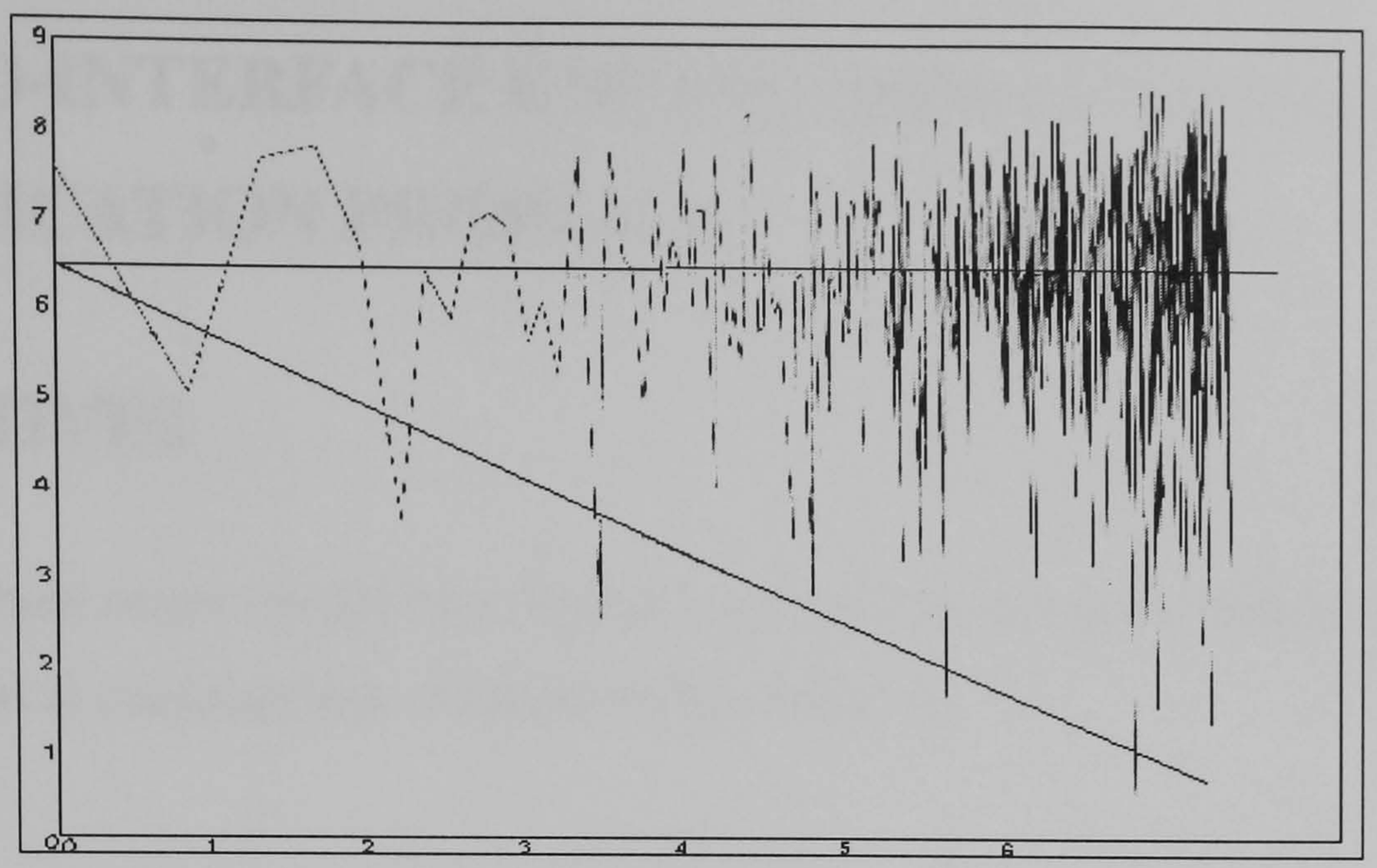


Figure 8.4
log (power) verses log (frequency)
gradient = 0 (white noise)

This Chapter introduced a digital communication system that employs fractal modulation for secure transmission. Various aspects of modelling noise, and details of computing fractal noise were then discussed. In the following Chapter, fractal noise is used to develop adaptive neuro models to optimise the digital signal communication system's performance

9 A NEURO-INTERFACE ENGINE FOR A DIGITAL COMMUNICATION PROBLEM

9.1 OBJECTIVES

To develop a robust neuro-model for a digital communication system that encodes bit streams into a format that is characteristic of transmission noise, to:

1. overcome the system's error rates generated with low signal/noise ratios, and
2. optimise the decoding process in terms of speed through reduced computation, and robustness of technique in presence of various types of unwanted signals.

9.2 DETAILS

A digital communication system that codes binary data in a form indistinguishable from the background noise is considered (see Chapter 8). The system, uses fractal modulation for digital transmission and reception of sensitive information. That is, instead of transmitting a frequency modulated signal, in which 0s and 1s are allocated different frequencies, a fractal signal is transmitted in which 0s and 1s are allocated different fractal dimensions. Basic steps include:

(a) Given a binary sequence, dimensions $D_{\min}$ and $D_{\max}$ are allocated to bit 0 and bit 1 respectively.

(b) Computing fractal signal of length N for each bit in sequence.

(c) Concatenating results to produce a continuous stream of fractal noise n_i.

On receiving the composed fractal signal (Figure 9.1):

(d) Fractal demodulation takes place - this process gives the array of D values that describe how the fractal dimension changes over the input signal.

(e) The encoded binary information is recovered from the array of D values.

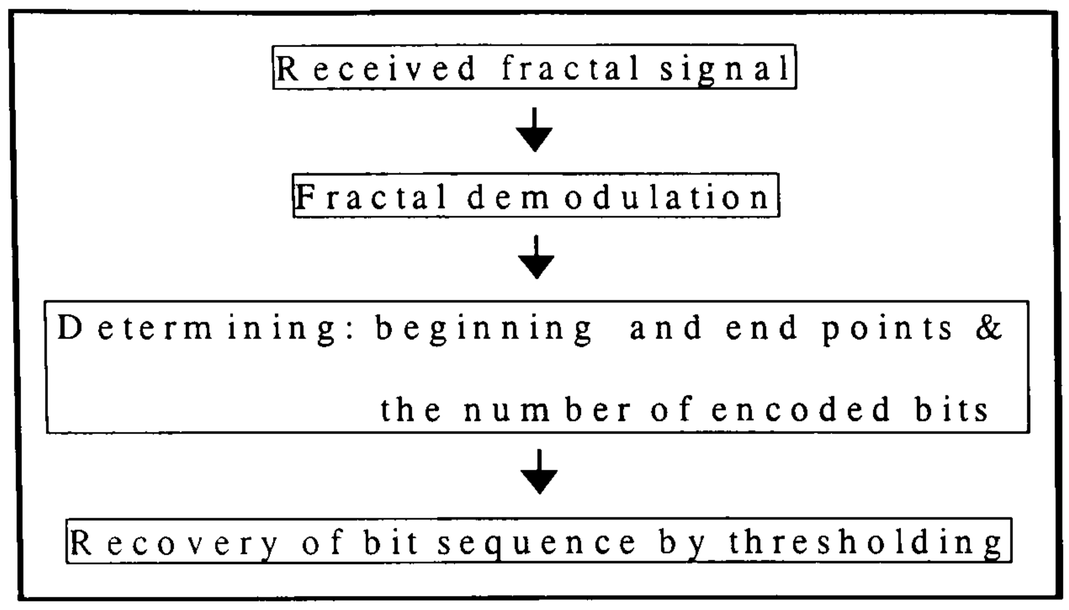


Figure 9.1
On receiving the composed fractal signal

Implementation of (e) leads to:

- determining the beginning and end points of transmission, and the number of bits
- recovery of bit stream using the following algorithm:

Given

$$\Delta = D_{\min} + \frac{D_{\max} - D_{\min}}{2}$$

If

$$D_i \leq \Delta \quad then\, bit = 0$$
$$D_i > \Delta \quad then\, bit = 1$$

The reconstruction algorithm, yields low error rates with a relatively high level of noise provided that:

(i) the difference in fractal dimension is not too small and

(ii) many fractals per bit are used.

9.3 A NEURO-MODEL

Based on (i) and (ii) being observed, a relatively flexible and efficient neuro-filter is proposed to replace the thresholding approach of recovering the binary information. Combined with the decoding technique, adequately trained networks can optimise recovery of the encoded binary information from the array of D values. Additionally, such models can optimise the overall decoding process in terms of :

(a) speed, through reduced computation (once training is complete), and

(b) robustness of technique in presence of various types of unwanted signals.

9.3.1 Data

- **2 Fractal noise convolution**

  - Fractal noise is a good model for most types of noise found in nature (Mandelbrot, 1977 and 1983). This includes the types of noise encountered in transmission.

  - Most types of transmission noise are 'statistically self-affine'. This term refers to random processes that have similar probability density functions at different scales. Thus when zooming into a random fractal signal, although the pattern of amplitude fluctuations changes across the field of view, the distribution of these amplitudes remains the same.

  - It was thought that convolution of fractal noise with bit streams would provide adequate information for developing a robust neuro-model

  - Using '$1/f^{q}$ power spectra with the value of exponent 'q' between 0.0 and 1.5, noise with the following description was produced:

(a) white noise,

(b) non-specific coloured noise,

(c) $1/f$ fractional – Pink noise,

(d) anti-persistent fractional Brown function,

(e) non-persistent fractional Brown function – Brown noise, and

(f) persistent fractional Brown function.

* Remark - random number used: an N by N matrix, with random entries chosen from a normal distribution with mean zero and variance one.

Data sets for the paradigms were then developed through convolution of binary sequences, up to 1000 bits, with the noise produced (sample data files are included with the accompanying disk). Figures 9.2 (a) and 9.2 (b) display examples of a variety of noises generated for convolution. They also show the following binary sequence (Table 9.1) convoluted with pink 9.2 (a), brown and white noise 9.2 (b).

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Table 9.1 Sample data

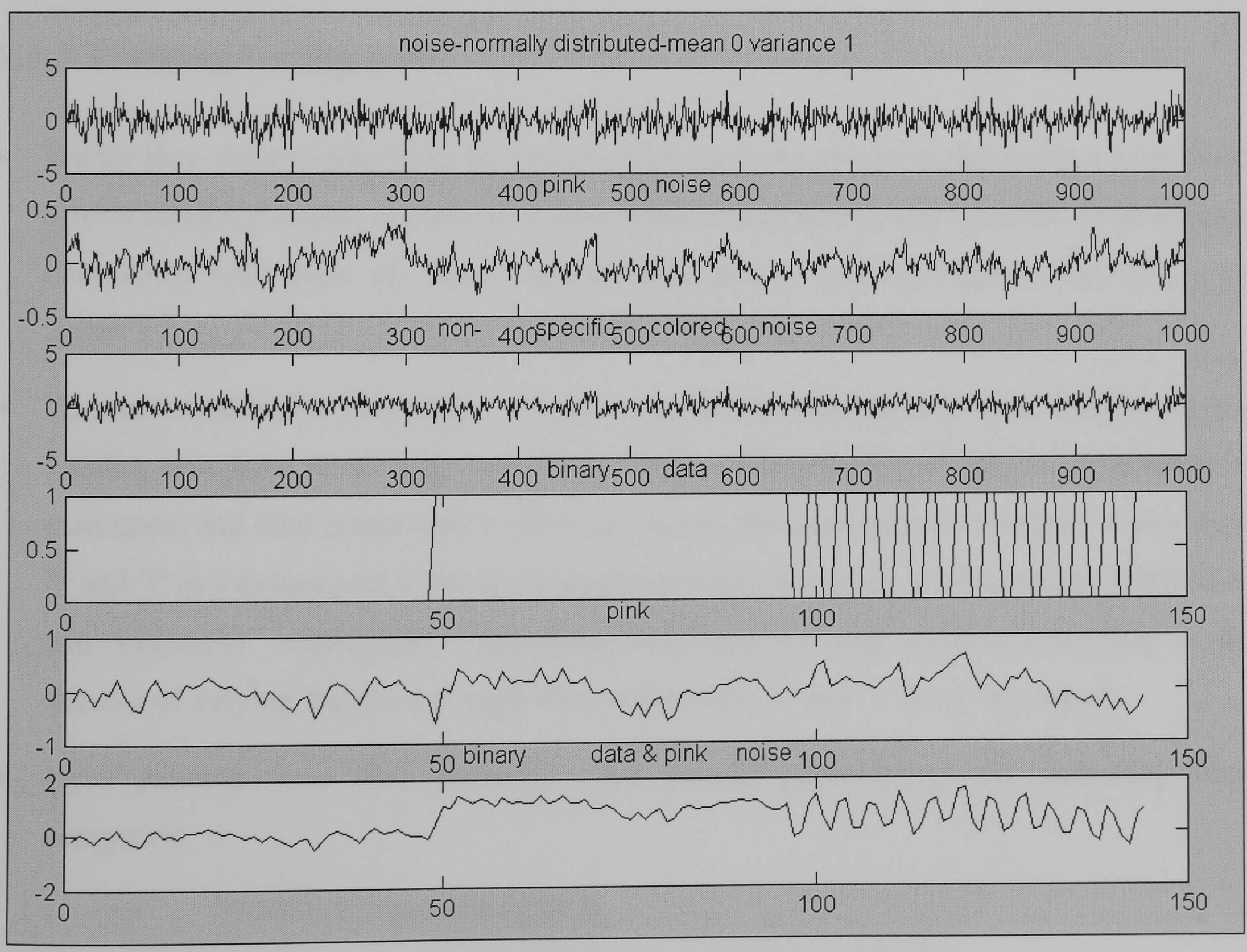


Figure 9.2 (a)
Sample of a variety of noise generated, binary data, and binary data convolved with pink noise

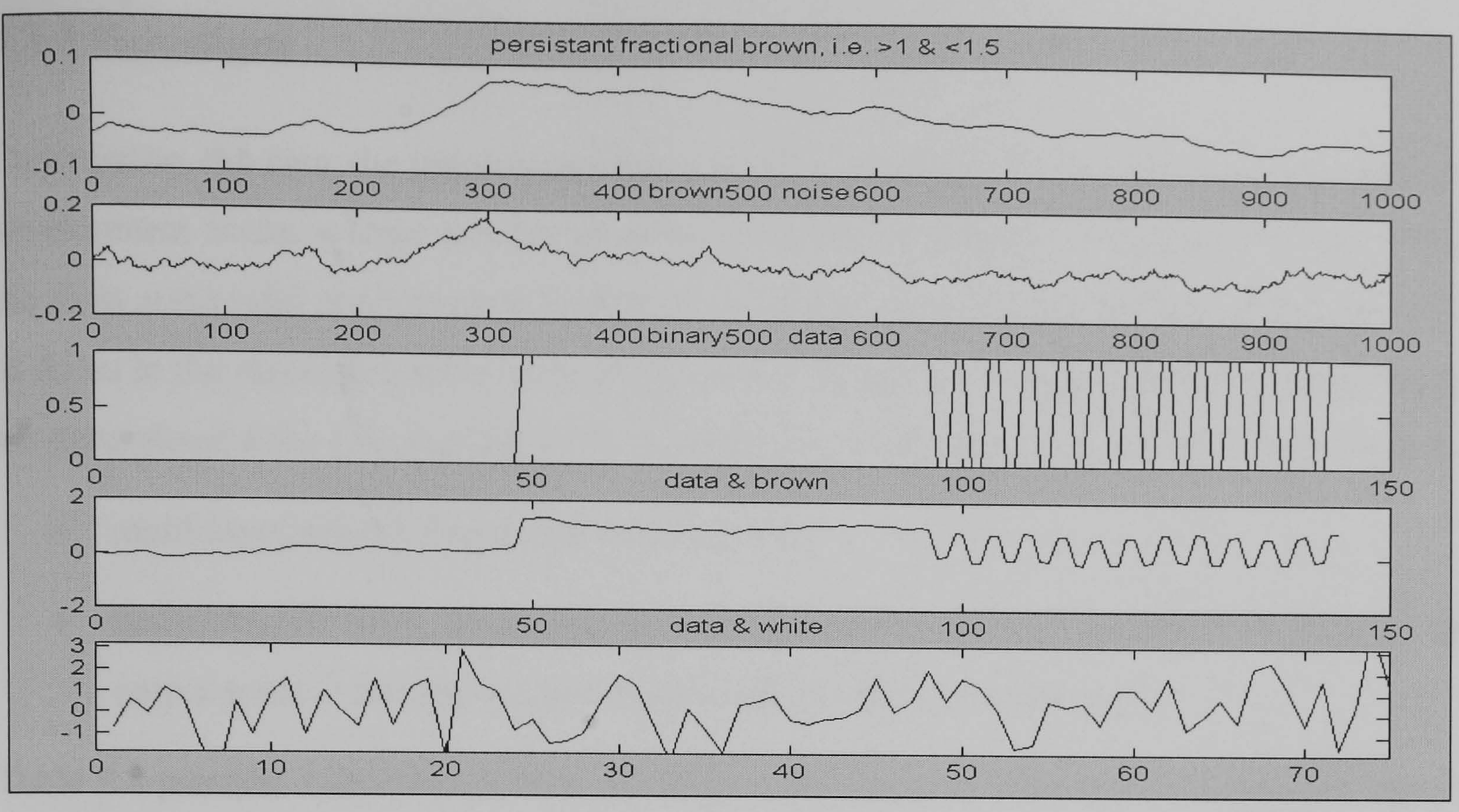


Figure 9.2 (b)
Sample noise, binary data, and data convolved with brown and whit noise

9.3.2 Training/testing sets

- Rather than determining what bit was encoded in every part of signal using a predefined thresholding technique, models were developed using training sets made-up of bit streams convoluted with noise of various descriptions, and of varying degrees, e.g. 20% pink noise.

- To accommodate for the number of fractal signals representing one bit, or fractals per bit, training sets were developed, with their inputs consisting of each bit being repeated at least once, and then convoluted with fractal noise. For example, if repeated 7 times, then '0' and '1' in a training set, prior to noise convolution, would be represented by '00000000' and '11111111' respectively. Based on this, input training patterns consisting of bit streams of varying length, e.g. eight bits, to represent '0' and '1' were developed.

- Input patterns were then extended for broadly representing the following three categories:

  (i) fractal bits representing bit 0

  (ii) fractal bits representing bit 1

  (iii) combinations of bits representing code determined by the transmission protocol, e.g. to find beginning and end of transmission.

9.3.3 Paradigm

Considering, the data, the training procedure most appropriate to the application and software development issues a large number of networks were developed. The objective was to find the most successful configuration to reach a minimum value for the average RMS error, and to do so in the shortest number of training cycles. Based on a large number of experiments, the generalised delta rule together with the following most efficient structures were adapted:

- multi-layered fully connected feed-forward
- multi-layered fully connected feed-forward with extended feedback of middle and output units to themselves and to other units within the same layer.

Table 9.2 presents examples of these paradigms producing no errors, when tested on unseen data. The networks were trained on binary data convoluted with fractal noise. Random noise of up to 20% (with a set decay rate) was additionally applied to each input, every time it was presented to the network. The noise was calculated as a pseudo random from a flat distribution with its maximum amplitude equal to the value chosen. For example for a noise value of 20%, each input bit had a value between -/+ 0.2 added to it independently. This causes the network to generalise, as it only sees a series of approximations of the input.

The results in Table 9.2 show that training took longer in the fully connected feed-forward network with no feedback. Networks with feedback, because of their instability problem, are generally more difficult to train. In an stable network, successive iterations produces smaller and smaller output changes until eventually the outputs become constant. In an unstable network, the process may never end. Chaotic systems are one example of unstable networks. For further comparison, artificial neural networks outputs displaying the testing results of the networks in Table 9.2 are presented in Figure 9.3. They show:

- the feedforward networks outputs (a) and (b), after reaching a 'max output unit error' of <o.2 after 281 cycles in (a), and < 0.1 after 541 cycles in (b), as well as
- the feedback networks outputs (c) and (d), after reaching a 'max output unit error' of <o.2 after only 59 cycles in (c), and < 0.1 after 105 cycles in (d) respectively.

Networks / **Specifications**	Multi-layered **fully-connected feedforward** with **backpropagation**	Multi-layered **fully-connected feedforward** with **extended feedback** of middle and output units to themselves and to others within the same layer with **backpropagation**
Hidden Layers	**2**	**1**
Connections	**32**	**19**
Dimensions	input 4 – middle 3 & 2 – output 3 PEs	input 4 – middle 2 – output 3 PEs
Initialise	**± 1%**	**± 1%**
Learning rate	**0.1**	**0.1**
Momentum term	**0.9**	**0.9**
Maximum output unit error of < 0.2	achieved after **281 cycles** of training	achieved after **59 cycles** of training
Maximum output unit error of < 0.1	achieved after **541 cycles** of training	achieved after **105 cycles** of training
Training (on unseen binary data convolved with fractal noise)	on binary data with **each bit repeated** before convolution with fractal noise (-to represent fractals/bit)	on binary data with **each bit repeated** before convolution with fractal noise (-to represent fractals/bit)
Extra training noise added	± 20%	± 20%
Decay	0.0005	0.0002 adjusted to 0.02 after 59 training cycles
Testing (on unseen binary data convolved with fractal noise)	**No error** (if each training bit was repeated a minimum of 3 times)	**No error** (if each training bit was repeated a minimum of 3 times)

Table 9.2
Examples of paradigms producing no errors

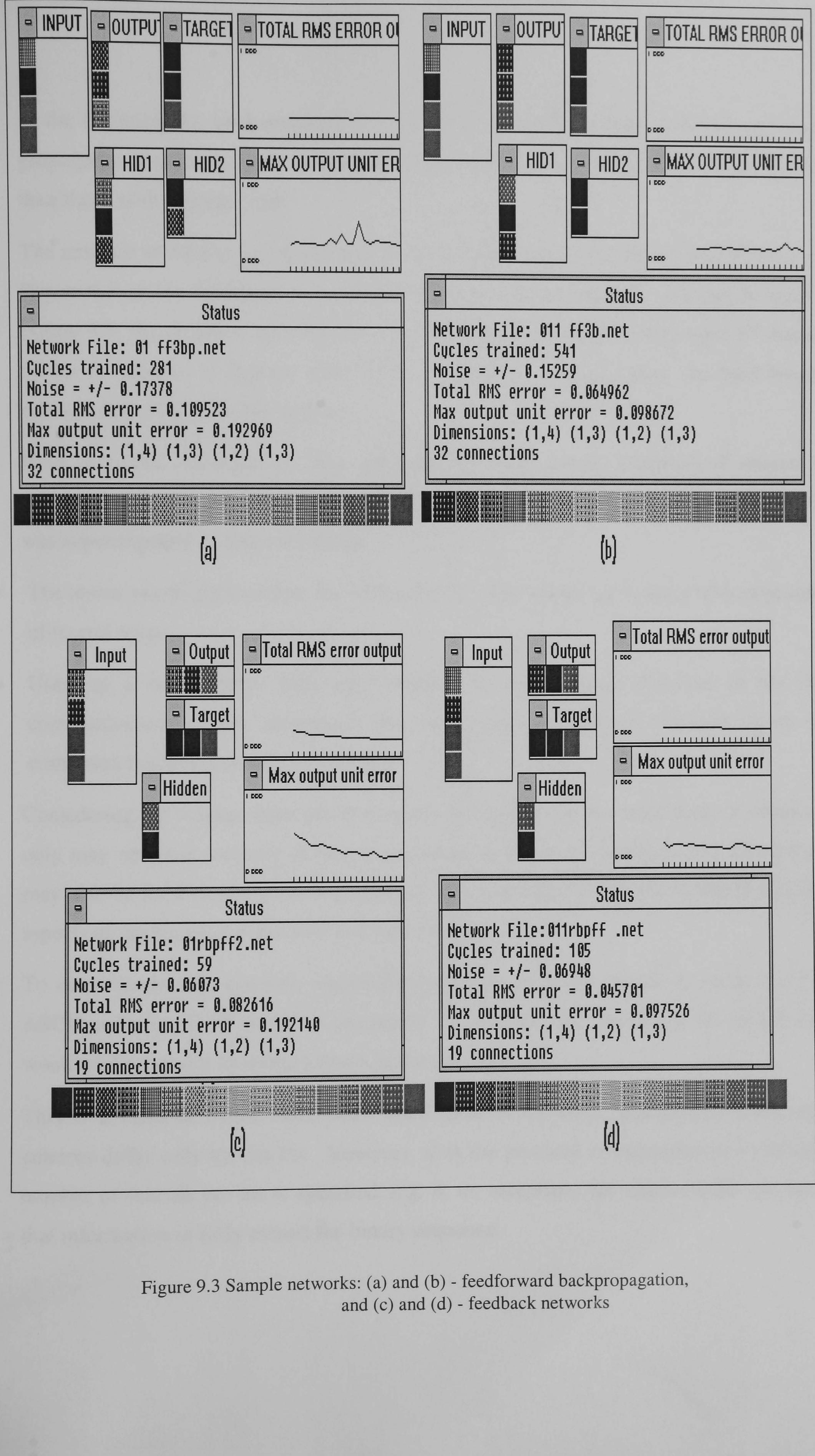


Figure 9.3 Sample networks: (a) and (b) - feedforward backpropagation, and (c) and (d) - feedback networks

- In the experiments, back-propagation networks, i.e. feed-forward networks with back-propagation algorithm, with two middle layers generally provided better generalisation than those with a single layer.

- The network of choice, as highlighted in the sample experiments shown in Table 9.2 and Figure 9.3, is the feedback or recurrent back-propagation network. As can be seen in Figure 9.3, the feedback network has only 19 connections, and it only takes 59 training cycles to achieve a slightly better level of training accuracy than the feed-forward network in 9.3 (b) with 541 cycles.

- Both networks, managed to fully recognise test data which consisted of unseen bit streams convoluted with fractal noise. Additionally, an extra noise level of up to +/- 0.2 was superimposed on them at random.

- The above results demonstrate the efficiency of robust feedback models with experience of fractal noise.

- Used as a neuro filter, they may replace the thresholding function in the data communication system discussed, recovering the original bit sequence from the composed fractal signal more efficiently.

- Considering the transmission protocol when developing the network, such a model not only may optimise recovery of binary sequences in terms of efficiency and speed, but it may also be used to recognise sequences of bits, making it easier for to identify various aspects of the signal, e.g. the start and stop points.

- To demonstrate this, consider transmitting a choice of 8 bits sequence, using the 8-bit ASCII code for all 95 printable characters. It was thought that using the ASCII code would make it easier to identify errors in the testing stage.

- This is a difficult neural task, even when there is no noise, particularly when input patterns differ only by one bit. However, if in the protocol of transmission a minimum number of fractals per bit is specified, e.g. 4, on reception, the neuro-model can utilise that information to fully extract the binary sequence.

i.e.

the input: *8-bit ASCII repeated 4 times.*

the output: *8-bit ASCII .*

- This is demonstrated by the neuro-model shown in Figure 9.4. The model took only 46 cycles to produce a total RMS error of < 0.06 with a maximum output unit error of ≈ 0.2.

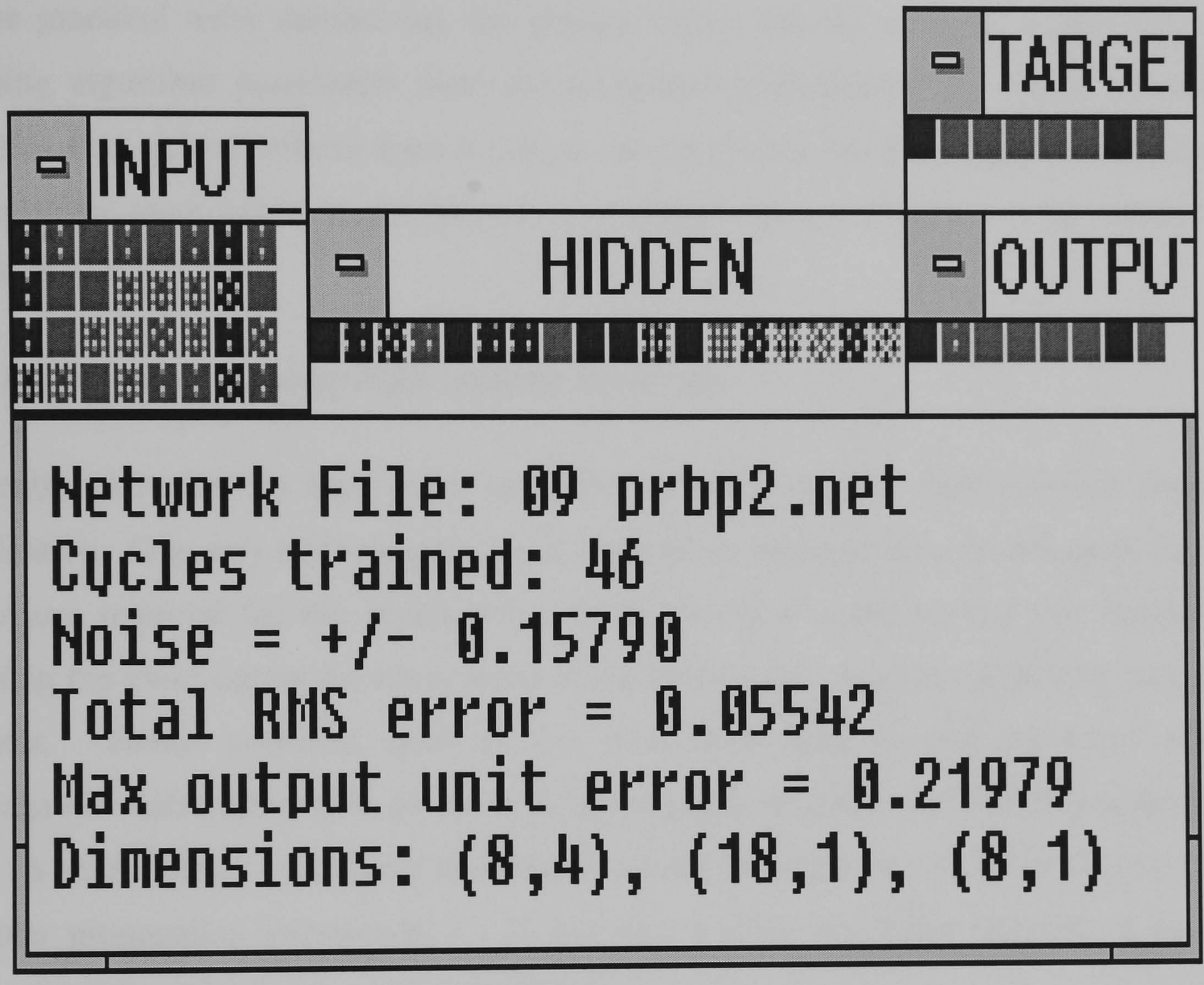


Figure 9.4
A back-propagation recurrent neuro-model

These results together with all the experimental results presented in this research work are further discussed in the following Section.

9.4 REFLECTING ON ALL EXPERIMENTAL RESULTS

The theory of neural networks is not yet sufficiently matured to allow a network application to be precisely designed and developed. The process involves trial-and-error as well as research and planning. It starts with determining the most appropriate network to implement. It is then followed by different aspects of network design such as deciding upon the network architecture and the learning algorithm parameters.

In the practical work carried out, the precise values for the network architecture and the learning algorithm parameters were all determined experimentally. Using demonstrative samples, this section reflects upon the experimental results, the most significant findings, and some of the main points in the process of deciding upon a suitable configuration for such applications.

9.4.1 The most appropriate neural network

Presently there are no hard rules available for choosing the right network for a given application. One way of finding the most appropriate network is to decide upon the training procedure required for the application. In the practical work carried out, because I was defining the exact output for every input in the training set, supervised training was the most suitable. Hence networks such as the multi-layer feed-forward network with back-propagation (also referred to as the back-propagation network) and the Boltzmann machine (and its refinements) were likely candidates capable of mapping and producing new outputs. Counter propagation (subsection 7.7.2) was also a likely candidate because in training the network receives the input and the target vector.

The general steps taken in the development of the neuro models were as follows:

- creating various network structures,
- performing training on data with/without noise superimposed on it (noise was used in order to accustom the networks to noisy data sets),
- halting the training process when the total RMS error reached less than 0.05, and the maximum output error reached less than 0.1,
- comparing networks in terms of efficiency and performance,
- preparing new data for testing,
- using unseen new data to perform testing in terms of recognition and generalisation,

- analysing and applying the results.

A large number of experiments, displaying the various aspects of the process were saved on hard disk for further analysis.

9.4.2 On using back-propagation

- The paradigm of choice, as it is for most pattern recognition applications, was eventually decided to be a multi-layered fully connected feed-forward network with back-propagation. Sample experimental results of the developed networks are provided in Table 7.2 and Table 9.2.

- As can be seen from Table 9.3, which is a reduced version of Table 7.2, back-propagation networks with two middle layers generally appeared to provide higher accuracy and better generalisation than those with a single middle layer. Network number 16, which has two hidden layers with similar specifications, managed to correctly recognise all the test patterns after only 618 cycles of training. When tested on unseen data sets with noise levels of 0.1, 0.15 and 0.2 added to them, the network managed to recognise 18/18, 17/18 and 16/18 respectively, suggesting a gradual deterioration in performance.

Networks / Specifications	12	13	14	16
Hidden Layers	1	2[similar]	1	2[similar]
Training Patterns	110[normalised & scaled]	110[normalised & scaled]	120[normalised & scaled]	120[normalised & scaled]
Cycles Trained	1139	358	1192	618
Testing[patterns recognised]	14 out of 18	15 out of 18	15 out of 18	18 out of 18

Table 9.3 Applying back-propagation to boundary cases. See also Table 7.2

- All the networks with 2 hidden layers used 6 processing elements in the first middle layer and 4 in the second. More middle (or hidden) layer processing elements generally

resulted in poor generalisation. That is the networks tended to memorise the training set rather than learn it. Defining too few hidden processing elements mostly resulted in incorrect classification.

9.4.3 Extending the back-propagation feed-forward with feedback

- For the multi-layered feed-forward networks in Table 9.3, I used The Generalised Delta Rule to perform static mappings of input data to output data. This class of networks have no dynamic memory as the response of the network depends on its current inputs and the values of synaptic weights. By allowing extensions in the form of feed-backs of hidden and output units to themselves and to other units in the same layer (see Figure 9.5) dynamic networks were developed. In such networks, the generated output depends both on the current inputs and on the current state of the units in layers with feedback connections. As discussed earlier in section 5.2, feedback networks can offer greater computational advantages over purely static neural networks. However, the theory of dynamic neural networks in general, is not well developed as compared to static networks.

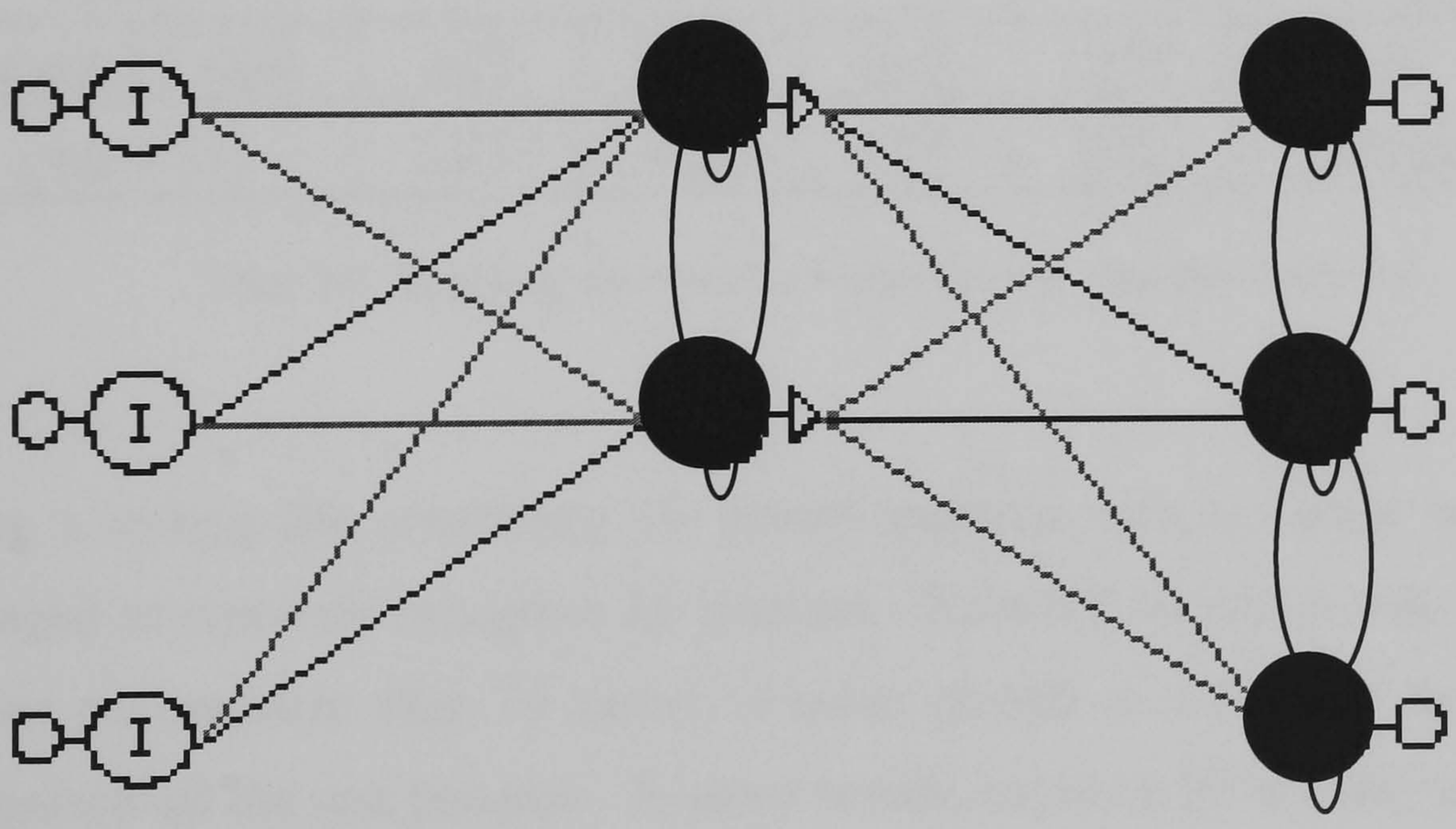


Figure 9.5

- To train the feedback network shown in Figure 9.5, I used Rumelhart's method (Rumelhart et al., 1986) to produce an equivalent static network. I then applied the

Generalised Delta Rule to train the network. The main drawback with unwinding a recurrent network in time to create an equivalent feedforward network is the memory resources required. For a further discussion on such networks, see Almedia (1987).

- By applying noise randomly to each input training pattern just before presenting it (each time), the network only saw a series of approximation of that input. The technique forces the network to generalise.

- Sample experiments applying the recurrent back-propagation networks to the experimental data within a noisy environment are included in Table 9.4. All the networks have the same configurations, but were trained using different noise levels. At the start of training, weights for the networks were initialised to values between -0.1 and +0.1.

Networks / Specifications	19	20	22	23	24	25
Training Data	120	120	120	120	120	120
Noise [+/-]	0.01	0.1	0.1	0.1	0.2	0.2
Decay	0.001	0.001	0.002	0.002	0.05	0.05
Cycles Trained	1962	1988	832	205	stopped 100000 local minimum	340
Testing[with 0.0 noise]	16/18	18/18	18/18	18/18	not applicable	16/18
[with -/+0.1]	13/18	18/18	18/18	18/18		15/18

Table 9.4 Applying recurrent back-propagation. See also Table 7.3

- Using a testing file containing 18 unseen patterns with no noise added, network 19 managed to correctly recognise 16 patterns. Network 20 which was trained in a much noisier environment than 19 (level of noise started at 0.1 rather than 0.01) correctly recognised all the test patterns. In other words, network 20 actually used the additional noise to improve its performance.

- To reduce the number of training cycles of network 20, the added noise was reduced in bigger chunks in network 22, which is otherwise similar to 20. That is a decay rate of 0.02 rather than 0.01 was used for 22. As a result the number of training cycles for 22 was reduced to less than half of that of 20 (from 1988 to 832). By increasing the stopping criteria (i.e. the maximum output unit error value) from 0.1 to 0.2, network 23 ,which is otherwise the same as 22, managed to correctly recognise all the test patterns after only

205 cycles of training. Network 25 started training with a significant amount of noise (0.2), which makes patterns difficult to recognise for anybody. Noise was then reduced in much larger quantities than that of 23 (i.e. 0.05 rather than 0.002). The network managed to recognise 16 patterns after 340 cycles of training.

- It was generally experienced that a large amount of noise (e.g. more than 0.15) and/or reduction of noise in huge chunks caused instability. That is training in such an environment did not necessarily produced smaller and smaller output changes. For example, training for network 24 which is exactly the same as 25 was halted after 100,000 cycles, as it was clearly stuck in a local minimum.

- When tested on unseen patterns with a noise value between -0.1 and +0.1 superimposed on them at random, correctly trained networks (e.g. 23) could recognise all the patterns. However, networks generally showed a degradation in performance with any increase in the noise level. For example, network 25 which recognised 16 out of 18 patterns with no noise added, only recognised 15 patterns, when a random noise between -0.1 and +0.1 was added to the test patterns. On increasing the amount of noise to -/+ 0.15, the network only managed to recognise 14 patterns.

- Overall, adding random noise greatly improved the performance of the networks both in terms of training and generalisation abilities. The technique forced the model to train on a series of approximations of the input, resulting in greatly improved performance in terms of training and generalisation abilities.

9.4.4 A neuro fractal model

- Based on a large number of experiments, the main findings in 9.4.1, 9.4.2, and 9.4.3 were adapted in the development of a neuro interface engine for the digital communication application.

- The most significant findings were:

  1. Dynamic networks were particularly appropriate for such applications.
  2. The performance of the neuro fractal models greatly improved when random noise was used.

- To accustom the networks with transmission noise, fractal noise was used in training. Additionally random noise was added to data sets, and then gradually reduced.

- Fractal noise is a good model for most types of noise, including transmission noise.
- Most types of transmission noise are ''statistically self-affine', i.e. random with similar probability density functions at different scales.
- Fractal noise superimposed on binary sequences provided the training data. Figure 9.5 Shows an example of a data set with added

  25% noise with normal distribution, mean 0, and variance 1.

  Pink noise, and

  Brown noise.

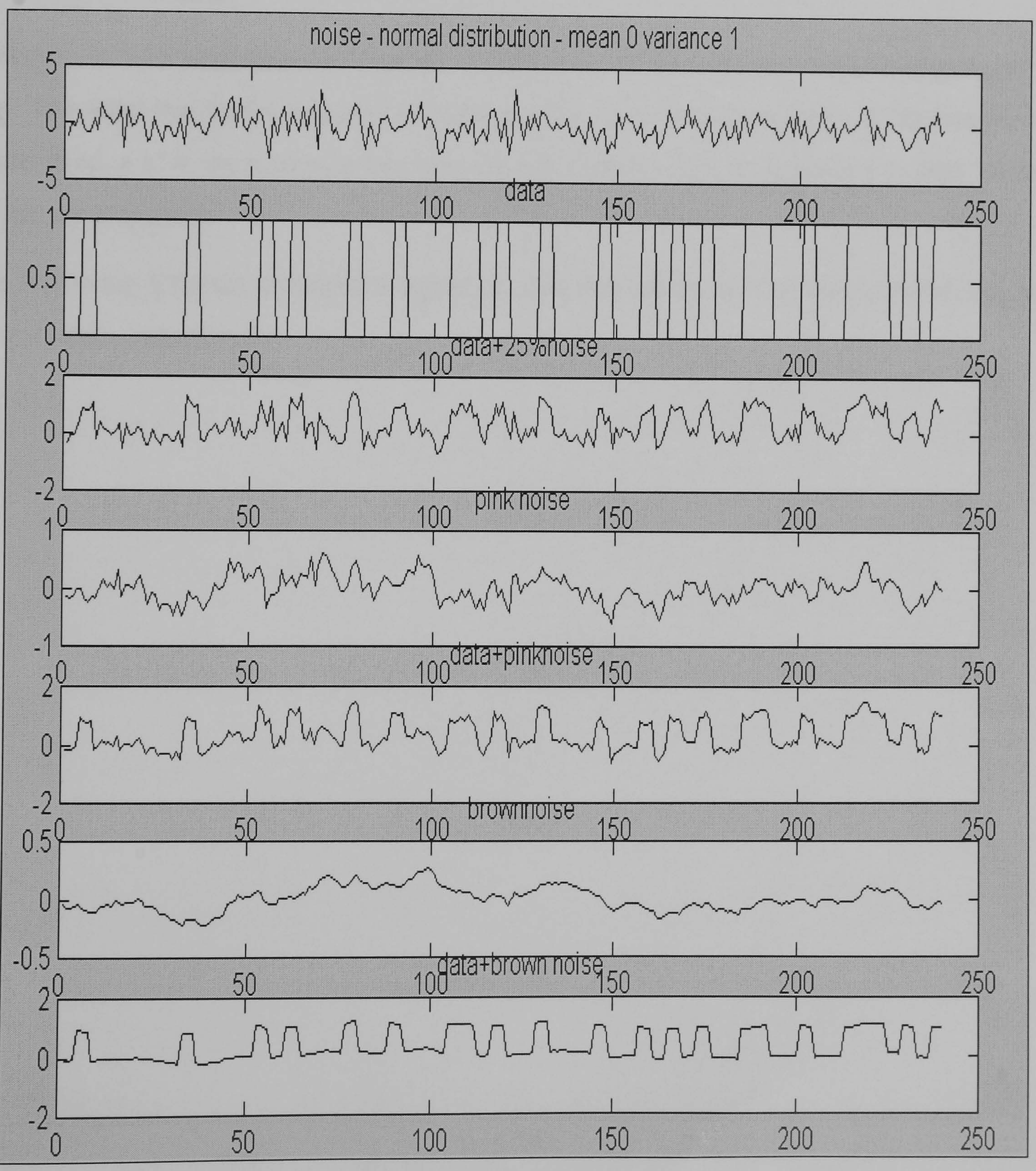


Figure 9.5
Noise convolution for develop training data sets

- To accommodate for the number of fractal signals representing one bit, training sets consisted of each bit being repeated at least once, prior to noise convolution.
- The paradigm of choice was the recurrent back-propagation model. It had only 19 connections, and took 59 cycles to be able to fully recognise unseen bit streams with fractal noise.
- Using the 8-bit ASCII code for all 95 printable characters, the developed model was then considered for recognition of sequences of bits. Such a scheme is a difficult task even when there is no noise.
- However, it was demonstrated that the model can be utilised to recognise sequences of bits. Provided that in the protocol of transmission, a minimum number of fractals per bit is specified, e.g. 4, on reception the neuro-model can use that information to fully extract the binary sequence.

The following Chapter provides a summary and discussion of the work carried out, and the literature considered.

10 CONCLUSIONS AND FURTHER RESEARCH

10.1 OVERVIEW

Artificial neural networks and fractal geometry promise great suitability for digital signal communication particularly in dealing with unwanted signals. This project aimed at investigating and practically analysing such promise.

To accomplish this the following objectives were set:

1. To review signal processing techniques (Chapter 2), fractal geometry (Chapter 3), and the fundamental concepts of artificial neural network technology (Chapters 4 and 5).

2. To investigate any link between artificial neural networks and fractal geometry (Chapter 6).

3. To employ a pattern recognition problem to explore the issues regarding incorporation of approximate models, mapping, and generalisation attributes of neuro-paradigms in presence of noise (Chapter 7).

4. To develop a robust neuro-model for a digital communication system that employs fractal modulation for secure transmission (Chapters 8 and 9), to

    - overcome the system's error rates generated with low signal/noise ratios, and

    - optimise the decoding process in terms of speed through reduced computation, and robustness of technique in presence of various types of unwanted signals.

These objectives have been achieved.

10.2 MAIN FINDINGS OF THIS PROJECT

The main revelations and contributions of this work are as follows:

1. The whole process of applying static models to a real world pattern recognition task, in a noisy environment has been experimentally analysed. (Chapter 7, Table 7.1). Such models are inherently stable as no information is fed back during operation.

2. By extending the static networks in Table 7.1 with feedback connections, the utility of dynamic networks in neuro-control applications were practically examined (see, Chapter 7, Table 7.2). When conducting these experiments, it was found that training the networks with feedback is more problematic (e.g. in terms of stability) than those presented in Table 7.1. As discussed earlier in 5.2, and 9.4.3, recurrent networks have dynamic memory and offer great potential for further research efforts, particularly in the area of developing neuro models to learn relationships between successive vectors, if such relationships exist.

3. In the process control benchmark, to accustom the neuro models to noisy data present in many experimental situations, noise was applied to each input training pattern just before presenting it (each time). The performance of the developed models, were then measured on the basis of their ability to respond to previously unseen data. On testing, it was generally found that:

- A large amount of noise (e.g. 0.2) and/or reduction of noise in huge chunks may cause instability, that is training did not necessarily produce decreasingly smaller output changes (see Table 7.3 network 24).
- Networks developed on the basis of their training set only (i.e. with no noise added) generally did not give satisfactory results when tested on new data.
- Careful use of noisy data effectively improved the network's performance in terms of generalisation and learning as well as causing a significant reduction in the number of training cycles (see Table 7.2and Table 9.2). In other words, the added noise actually helped the networks in their pattern recognition task.
- **Why**
- Applying a significant amount of noise, to each element of all the input training vectors independently, and reducing the amount gradually, forced the networks to train on a series of approximations of the input patterns. This drove the networks to generalise.
- The performance of the models developed, were then measured on the basis of their generalisation ability or their ability to respond to previously unseen data.
- Because the networks were in effect trained on an unlimited source of training samples (provided by adding random noise to the available training vectors), the method was effective in improving model performance for both applications.

- A different approach to study the effects of adding noise to the inputs is to look at the models determined on the basis of their training sets with no random noise superimposed on them. Those networks often suffered from the problem of generalisation. That is they often did not give satisfactory results when tested on new data. This problem is similar to that of model selection in the context of data modelling.

- The common approach to solve the problem is to make sure that the training set is large and fully representative, covering boundary cases. Another approach is regularisation, but this method is problem dependent, and it may not be clear what type of regularisation to use prior (Poggio and Girosi, 1990).

4. Using the practical findings on robust neuro models and using fractal analysis, based on

$$\frac{d^{\,q(t)}}{d\,t^{\,q(t)}} f(t) = n(t)$$

n (t) – White noise

f (t) - Fractal time series

q(t) - Fractal dimension signature

$$1 < q(t) < 2 \qquad \forall t$$

models were developed for a digital signal communication system that uses fractal modulation for secure transmission. The practical work carried out demonstrated the efficiency of robust neuro models with experience of fractal noise, in the recovery of the original bit sequences. To accustom the networks with transmission noise, fractal noise was used in training because of its following behaviour:

-independent of size of problem

-self similar at all scales.

Additionally, random noise was added to data sets, and then gradually reduced to force the networks to train on a series of approximations of the input patterns. This drove the networks to generalise, resulting in an improvement in their performance.

A reasonable conclusion to be drawn from the experimental work carried out, as well as the literature surveyed, would recommend fractal noise, and digital signal communication using fractal modulation as a promising area for further analysis and development of robust neural systems. With most types of transmission noise being 'statistically self-affine', using fractal

noise, neuro models can be developed that perform better than conventional approaches in dealing with noise fields. Conventional approaches don't perform well, as the physical origins of many noise types are not well understood.

10.3 DISCUSSION AND FUTURE RESEARCH

The general focus of this research was to analyse the suitability of fractal geometry and adaptive neuro-models to deal with unwanted signals in signal communication applications, optimising conventional algorithms. It was concluded that a robust neuro-model can overcome error rates generated with low signal/noise ratios. Using fractal noise convolution, the two main models developed were feed-forward and feedback networks.

10.3.1 Static models

It is well established that multi-layer feed-forward networks can approximate non-linear functions to any desired degree of accuracy. They have been applied successfully in modelling, but they suffer from many limitations including:-

- the very slow learning rate- attempts to accelerate the learning process by increasing the values of the learning algorithms' gains (constants), generally result in unstable systems,
- having no dynamic memory- their response depends solely on the current inputs and the values of the weights,
- many practical questions remain unanswered- for example the number of middle layer processing elements, or the relationship between the accuracy of the function being approximated with the number of hidden layers.

10.3.2 Dynamic models

Feedback or dynamic neural networks are networks with feedback connections. The feedback causes the network to have local memory characteristics. Like many of the real world control systems requiring to be modelled (e.g. the forward or inverse dynamic of a process) these networks are non-linear dynamical systems. They offer great computational

advantages over feed-forward networks. But the general theory with regard to architecture and learning algorithms has yet to be developed.

Neural networks with feedback are particularly appropriate for system modelling and identification. The development of better feedback models could result in significant progress in adaptive modelling applications.

10.3.3 Other structures

It is impossible to address all potentially applicable models, however, a review of the literature suggests the following models as currently popular with researchers.

- Adaptive fuzzy systems or fuzzy-neural networks - these systems (Section 5.3), are believed to have considerable potential in pattern recognition (Lawrence and Harris, 1993).

- The Cerebellar Model Articulation Controller (CMAC) Miller et al., 1990; Kraft and Campagna, 1990, Albus (1975) - this is a memory management technique in which fast computing is achieved by referring to a table rather than by solution of analytic equations.

- Professor Jerry M. Mendel, of Signal and Image Processing Institute, University of Southern California, in 'Uncertainty in fuzzy logic systems (to be published in Autumn 2000), reports on Fuzzy Logic II systems. Such systems, unlike conventional fuzzy logic systems claim effective modelling and handling of uncertainty. The system has been applied to chaotic time series when measurements are corrupted by additive noise, but they have not yet compared their results with neural models.

APPENDIX A: SAMPLE DATA SETS

This Appendix provides sample training / testing data sets for the work reported in Chapter 7. It consists of two sections:

- Section A provides sample fluid flow patterns of a full training set containing 55 vectors in their original digitised form (A1), as well as pre-processed, i.e. normalised and scaled down versions (A2).
- Section B provides sample data for the experiments in Chapters 8 and 9. Data sets include: a binary sequence used in the experiments (Section B1), noise (Section B2), pink noise (Section B3), brown noise (Section B4).

SECTION A – SAMPLE DATA USED IN CHAPTER 7

Section A1 'Training file containing input/ output patterns'

A training file containing 55 input/output patterns used for the experiments in Chapter 7.

TRAIN 13, 1, 3, 1

/* 55 I/O VECTORS */

/* Input vector 1 : */

1.100000, 1.440000, 1.480000, 1.450000, 1.460000, 1.450000, 1.450000, 1.490000, 1.470000, 1.490000, 1.460000, 1.480000, 1.480000,

/* Output vector 1 : */

0.500000, -0.500000, -0.500000,

/* Input vector 2 : */

0.610000, 1.220000, 1.190000, 1.290000, 1.280000, 1.300000, 1.290000, 1.300000, 1.320000, 1.300000, 1.330000, 1.320000, 1.300000,

/* Output vector 2 : */

0.500000, -0.500000, -0.500000,

/* Input vector 3 : */

0.000000, 1.520000, 1.220000, 1.420000, 1.410000, 1.420000, 1.420000, 1.450000, 1.450000, 1.440000, 1.420000, 1.450000, 1.470000,

/* Output vector 3 : */

0.500000, -0.500000, -0.500000,

/*Input vector 4 : */

0.060000, 1.250000, 1.470000, 1.470000, 1.490000, 1.460000, 1.330000, 1.280000, 1.260000, 1.390000, 1.440000, 1.470000, 1.440000,

/* Output vector 4 : */

0.500000, -0.500000, -0.500000,

/*Input vector 5 : */

0.000000, 1.250000, 0.100000, 0.780000, 0.600000, 0.820000, 1.100000, 1.190000, 1.520000, 0.660000, 0.000000, 1.100000, 0.870000,

/*Output vector 5 : */

-0.500000, 0.500000, -0.500000,

/*Input vector 6 : */

0.000000, 0.140000, 0.620000, 0.550000, 0.560000, 0.570000, 0.580000, 0.590000, 0.580000, 0.580000, 0.570000, 0.602000, 0.600000,

/*Output vector 6 : */

0.500000, -0.500000, -0.500000,

/* Input vector 7 : */

0.200000, 0.430000, 0.380000, 0.340000, 0.380000, 0.410000, 0.410000, 0.390000, 0.390000, 0.380000, 0.390000, 0.390000, 0.390000,

/* Output vector 7 : */

0.500000, -0.500000, -0.500000,

/* Input vector 8 : */

0.400000, 1.230000, 1.180000, 1.140000, 1.180000, 1.210000, 1.210000, 1.190000, 1.190000, 1.180000, 1.190000, 1.190000, 1.190000,

/* Output vector 8 : */

0.500000, -0.500000, -0.500000,

/*Input vector 9 : */

0.200000, 1.430000, 1.380000, 1.340000, 1.380000, 1.410000, 1.410000, 1.390000, 1.390000, 1.380000, 1.390000, 0.620000, 0.300000,

/*Output vector 9 : */

-0.500000, -0.500000, 0.500000,

/* Input vector 10 : */

0.000000, 0.070000, 0.500000, 0.570000, 0.540000, 0.590000, 0.610000, 0.580000, 0.590000, 0.580000, 0.630000, 0.610000, 0.590000,

/*Output vector 10 : */

0.500000, -0.500000, -0.500000,

/*Input vector 11 : */

0.200000, 0.130000, 0.300000, 0.370000, 1.440000, 1.570000, 0.410000, 0.280000, 1.390000, 1.180000, 1.430000, 0.410000, 1.390000,

/*Output vector 11 : */

-0.500000, 0.500000, -0.500000,

/*Input vector 12 : */

0.000000, 0.080000, 0.530000, 0.580000, 0.590000, 0.550000, 0.530000, 0.630000, 0.600000, 0.590000, 0.600000, 0.620000, 0.630000,

/*Output vector 12 : */

0.500000, -0.500000, -0.500000,

/* Input vector 13 : */

0.200000, 1.320000, 1.420000, 1.220000, 1.210000, 1.220000, 1.220000, 1.250000, 1.250000, 1.240000, 0.620000, 0.750000, 0.690000,

/* Output vector 13 : */

-0.500000, -0.500000, 0.500000,

/*Input vector 14 : */

0.200000, 1.320000, 1.020000, 1.220000, 1.210000, 1.220000, 1.250000, 1.250000, 1.240000, 1.220000, 0.450000, 0.490000, 0.470000,

/* Output vector 14 : */

-0.500000, -0.500000, 0.500000,

/*Input vector 15 : */

0.400000, 1.520000, 1.220000, 1.420000, 1.410000, 1.420000, 1.420000, 1.450000, 1.450000, 1.440000, 0.820000, 0.360000, 0.390000,

/*Output vector 15 : */

-0.500000, -0.500000, 0.500000,

/*Input vector 16 : */

0.200000, 1.450000, 1.100000, 0.980000, 0.800000, 0.120000, 1.310000, 1.390000, 1.320000, 0.860000, 0.200000, 1.300000, 0.970000,

/*Output vector 16 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 17 : */

0.200000, 1.050000, 0.100000, 0.580000, 0.400000, 0.620000, 0.910000, 0.990000, 1.320000, 0.460000, 0.200000, 0.900000, 0.670000,

/* Output vector 17 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 18 : */

0.400000, 1.250000, 1.300000, 1.180000, 1.000000, 0.320000, 1.510000, 1.520000, 1.520000, 1.060000, 0.400000, 1.500000, 1.170000,

/* Output vector 18 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 19 : */

0.260000, 1.450000, 1.270000, 1.270000, 1.290000, 1.260000, 1.530000, 1.480000, 1.460000, 1.490000, 1.240000, 1.270000, 1.240000,

/* Output vector 19 : */

0.500000, -0.500000, -0.500000,

/*Input vector 20 : */

0.140000, 1.050000, 0.270000, 1.260000, 1.290000, 1.060000, 0.130000, 0.800000, 1.060000, 1.190000, 0.240000, 1.300000, 0.240000,

/* Output vector 20 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 21 : */

0.100000, 0.400000, 0.420000, 0.400000, 0.450000, 0.440000, 0.460000, 0.480000, 0.480000, 0.490000, 0.460000, 0.470000, 0.470000,

/* Output vector 21 : */

0.500000, -0.500000, -0.500000,

/* Input vector 22 : */

0.830000, 1.030000, 1.080000, 1.070000, 1.090000, 1.100000, 1.080000, 1.090000, 1.080000, 1.070000, 1.090000, 1.100000, 1.110000,

/* Output vector 22 : */

0.500000, -0.500000, -0.500000,

/* Input vector 23 : */

0.050000, 1.420000, 1.110000, 1.320000, 1.330000, 1.330000, 1.320000, 1.340000, 1.350000, 1.310000, 1.320000, 1.350000, 1.370000,

/* Output vector 23 : */

0.500000, -0.500000, -0.500000,

/*Input vector 24 : */

0.180000, 1.250000, 1.480000, 1.470000, 1.490000, 1.460000, 1.430000, 1.480000, 1.460000, 1.490000, 1.540000, 1.570000, 1.540000,

/* Output vector 24 : */

0.500000, -0.500000, -0.500000,

/*Input vector 25 : */

0.000000, 0.250000, 1.100000, 1.380000, 0.600000, 0.820000, 0.100000, 1.190000, 0.520000, 0.680000, 0.130000, 1.140000, 0.940000,

/*Output vector 25 : */

-0.500000, 0.500000, -0.500000,

/*Input vector 26 : */

0.000000, 0.240000, 0.720000, 1.450000, 1.460000, 1.470000, 1.480000, 1.490000, 1.490000, 1.450000, 1.470000, 1.520000, 1.550000,

/*Output vector 26 : */

0.500000, -0.500000, -0.500000,

/* Input vector 27 : */

0.100000, 0.490000, 0.290000, 0.270000, 0.290000, 0.310000, 0.310000, 0.270000, 0.270000, 0.280000, 0.280000, 0.280000, 0.280000,

/* Output vector 27 : */

0.500000, -0.500000, -0.500000,

/* Input vector 28 : */

0.630000, 0.930000, 0.980000, 0.940000, 0.980000, 0.940000, 0.910000, 0.950000, 0.950000, 0.980000, 0.960000, 0.960000, 0.960000,

/* Output vector 28 : */

0.500000, -0.500000, -0.500000,

/*Input vector 29 : */

0.330000, 1.210000, 1.250000, 1.250000, 1.290000, 1.300000, 1.300000, 1.270000, 1.280000, 1.280000, 1.280000, 0.530000, 0.500000,

/*Output vector 29 : */

-0.500000, -0.500000, 0.500000,

/* Input vector 30 : */

0.000000, 0.040000, 0.600000, 0.570000, 0.550000, 0.570000, 0.620000, 0.590000, 0.590000, 0.580000, 0.600000, 0.600000, 0.580000,

/*Output vector 30 : */

0.500000, -0.500000, -0.500000,

/*Input vector 31 : */

0.000000, 0.790000, 0.300000, 0.470000, 1.430000, 1.880000, 1.410000, 0.380000, 1.240000, 1.450000, 0.430000, 1.410000, 1.390000,

/*Output vector 31 : */

-0.500000, 0.500000, -0.500000,

/*Input vector 32 : */

0.060000, 0.180000, 0.620000, 0.600000, 0.570000, 0.560000, 0.530000, 0.640000, 0.630000, 0.580000, 0.610000, 0.630000, 0.610000,

/*Output vector 32 : */

0.500000, -0.500000, -0.500000,

/* Input vector 33 : */

0.360000, 1.160000, 1.320000, 1.720000, 1.530000, 1.490000, 1.540000, 1.480000, 1.560000, 1.550000, 0.720000, 0.730000, 0.690000,

/* Output vector 33 : */

-0.500000, -0.500000, 0.500000,

/*Input vector 34 : */

0.220000, 1.220000, 1.320000, 1.520000, 1.510000, 1.520000, 1.550000, 1.550000, 1.540000, 1.520000, 0.350000, 0.360000, 0.330000,

/* Output vector 34 : */

-0.500000, -0.500000, 0.500000,

/*Input vector 35 : */

0.400000, 1.020000, 0.620000, 0.820000, 0.810000, 0.820000, 0.820000, 0.850000, 0.850000, 0.840000, 1.520000, 1.560000, 1.540000,

/*Output vector 35 : */

-0.500000, -0.500000, 0.500000,

/*Input vector 36 : */

0.200000, 0.940000, 1.200000, 0.600000, 0.790000, 0.430000, 1.510000, 0.390000, 1.580000, 1.560000, 0.500000, 0.200000, 0.970000,

/*Output vector 36 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 37 : */

0.330000, 1.060000, 0.110000, 0.460000, 0.500000, 0.890000, 0.810000, 0.690000, 1.520000, 1.460000, 0.520000, 0.290000, 0.970000,

/* Output vector 37 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 38 : */

0.400000, 0.670000, 1.290000, 1.170000, 1.150000, 1.310000, 1.520000, 1.190000, 1.530000, 1.160000, 0.450000, 1.160000, 1.590000,

/* Output vector 38 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 39 : */

0.650000, 0.950000, 0.640000, 0.630000, 0.610000, 0.560000, 0.530000, 0.580000, 0.560000, 0.570000, 0.640000, 0.660000, 0.620000,

/* Output vector 39 : */

0.500000, -0.500000, -0.500000,

/*Input vector 40 : */

0.400000, 1.040000, 0.270000, 1.270000, 0.790000, 1.170000, 0.240000, 0.800000, 1.000000, 1.230000, 1.440000, 0.700000, 0.980000,

/* Output vector 40 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 41 : */

0.000000, 0.070000, 0.500000, 0.570000, 0.540000, 0.590000, 0.610000, 0.580000, 0.590000, 0.580000, 0.630000, 0.610000, 0.590000,

/*Output vector 41 : */

0.500000, -0.500000, -0.500000,

/*Input vector 42 : */

0.000000, 0.069000, 0.630000, 1.480000, 1.490000, 1.500000, 1.500000, 1.410000, 1.480000, 1.490000, 1.410000, 1.420000, 1.430000,

/*Output vector 42 : */

0.500000, -0.500000, -0.500000,

/* Input vector 43 : */

0.070000, 0.920000, 1.520000, 1.420000, 1.790000, 1.720000, 1.520000, 1.550000, 1.550000, 1.540000, 0.320000, 0.350000, 0.380000,

/* Output vector 43 : */

-0.500000, -0.500000, 0.500000,

/*Input vector 44 : */

0.090000, 0.200000, 1.030000, 1.320000, 1.310000, 1.320000, 1.330000, 1.360000, 1.310000, 1.310000, 0.850000, 0.870000, 0.860000,

/* Output vector 44 : */

-0.500000, -0.500000, 0.500000,

/*Input vector 45 : */

0.000000, 1.020000, 1.420000, 0.890000, 0.880000, 0.870000, 0.870000, 0.870000, 0.860000, 0.890000, 0.830000, 0.220000, 0.200000,

/*Output vector 45 : */

-0.500000, -0.500000, 0.500000,

/*Input vector 46 : */

0.900000, 1.280000, 1.200000, 1.280000, 0.290000, 0.600000, 0.810000, 1.450000, 1.310000, 0.930000, 0.400000, 0.300000, 1.220000,

/*Output vector 46 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 47 : */

0.100000, 0.050000, 1.100000, 1.040000, 0.540000, 0.820000, 1.100000, 0.730000, 0.520000, 1.090000, 0.160000, 0.760000, 1.170000,

/* Output vector 47 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 48 : */

0.070000, 1.070000, 0.980000, 1.340000, 1.100000, 1.650000, 0.490000, 1.490000, 0.780000, 0.460000, 1.340000, 0.970000, 1.570000,

/* Output vector 48 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 49 : */

0.030000, 0.670000, 1.580000, 1.580000, 1.590000, 1.590000, 1.530000, 1.580000, 1.560000, 1.570000, 1.550000, 1.570000, 1.580000,

/* Output vector 49 : */

0.500000, -0.500000, -0.500000,

/*Input vector 50 : */

0.070000, 1.140000, 0.430000, 0.860000, 1.290000, 1.060000, 0.320000, 0.660000, 1.500000, 0.550000, 1.370000, 1.500000, 0.790000,

/* Output vector 50 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 51 : */

0.000000, 0.090000, 0.670000, 0.650000, 0.650000, 0.650000, 0.660000, 0.640000, 0.660000, 0.680000, 0.650000, 0.680000, 0.680000,

/* Output vector 51 : */

0.500000, -0.500000, -0.500000,

/* Input vector 52 : */

0.600000, 1.220000, 1.130000, 1.120000, 1.190000, 1.180000, 1.180000, 1.180000, 1.180000, 1.160000, 1.180000, 1.180000, 1.110000,

/* Output vector 52 : */

0.500000, -0.500000, -0.500000,

/* Input vector 53 : */

0.540000, 1.520000, 1.090000, 1.060000, 1.030000, 1.050000, 1.030000, 1.040000, 1.030000, 1.030000, 1.020000, 1.030000, 1.070000,

/* Output vector 53 : */

0.500000, -0.500000, -0.500000,

/*Input vector 54 : */

0.430000, 1.080000, 0.590000, 0.570000, 0.570000, 0.530000, 0.580000, 0.580000, 0.520000, 0.590000, 0.590000, 0.580000, 0.590000,

/* Output vector 54 : */

0.500000, -0.500000, -0.500000,

/*Input vector 55 : */

0.550000, 0.850000, 0.300000, 0.980000, 1.450000, 0.920000, 1.200000, 0.320000, 0.860000, 0.390000, 1.350000, 1.050000, 0.740000,

/*Output vector 55 : */

-0.500000, 0.500000, -0.500000,

END

Section A2 'Training file normalised and scaled down'

This sub section contains the normalised and scaled down (between -0.5 to +0.5) version of the training set provided in A1. This is done to remove the mean [i.e. offset the mean to zero] and to scale data within a specific range. Such an action was applied to all training and testing files to ensure compatibility with the computational methodology of the software package used.

TRAIN 13, 1, 3, 1

/* Input vector 1 : */

-0.500000, 0.371795, 0.474359, 0.397436, 0.423077, 0.397436, 0.397436, 0.500000, 0.448718, 0.500000, 0.423077, 0.474359,

0.474359,

/* Output vector 1 : */

0.500000, -0.500000, -0.500000,

/* Input vector 2 : */

-0.500000, 0.347222, 0.305556, 0.444444, 0.430555, 0.458333, 0.444444, 0.458333, 0.486111, 0.458333, 0.500000, 0.486111,
0.458333,

/* Output vector 2 : */

0.500000, -0.500000, -0.500000,

/* Input vector 3 : */

-0.500000, 0.500000, 0.302632, 0.434211, 0.427632, 0.434211, 0.434211, 0.453947, 0.453947, 0.447368, 0.434211, 0.453947,
0.467105,

/* Output vector 3 : */

0.500000, -0.500000, -0.500000,

/* Input vector 4 : */

-0.500000, 0.332168, 0.486014, 0.486014, 0.500000, 0.479021, 0.388112, 0.353147, 0.339161, 0.430070, 0.465035, 0.486014,
0.465035,

/* Output vector 4 : */

0.500000, -0.500000, -0.500000,

/* Input vector 5 : */

-0.500000, 0.322368, -0.434211, 0.013158, -0.105263, 0.039474, 0.223684, 0.282895, 0.500000, -0.065789, -0.500000, 0.223684,
0.072368,

/* Output vector 5 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 6 : */

-0.500000, -0.274194, 0.500000, 0.387097, 0.403226, 0.419355, 0.435484, 0.451613, 0.435484, 0.435484, 0.419355, 0.470968,
0.467742,

/* Output vector 6 : */

0.500000, -0.500000, -0.500000,

/* Input vector 7 : */

-0.500000, 0.500000, 0.282609, 0.108696, 0.282609, 0.413043, 0.413043, 0.326087, 0.326087, 0.282609, 0.326087, 0.326087,
0.326087,

/* Output vector 7 : */

0.500000, -0.500000, -0.500000,

/* Input vector 8 : */

-0.500000, 0.500000, 0.439759, 0.391566, 0.439759, 0.475904, 0.475904, 0.451807, 0.451807, 0.439759, 0.451807, 0.451807,
0.451807,

/* Output vector 8 : */

0.500000, -0.500000, -0.500000,

/* Input vector 9 : */

-0.500000, 0.500000, 0.459350, 0.426829, 0.459350, 0.483740, 0.483740, 0.467480, 0.467480, 0.459350, 0.467480, -0.158537,
-0.418699,

/* Output vector 9 : */

-0.500000, -0.500000, 0.500000,

/* Input vector 10 : */

-0.500000, -0.388889, 0.293651, 0.404762, 0.357143, 0.436508, 0.468254, 0.420635, 0.436508, 0.420635, 0.500000, 0.468254,
0.436508,

/* Output vector 10 : */

0.500000, -0.500000, -0.500000,

/* Input vector 11 : */

-0.451389, -0.500000, -0.381944, -0.333333, 0.409722, 0.500000, -0.305556, -0.395833, 0.375000, 0.229167, 0.402778, -0.305556,
0.375000,

/* Output vector 11 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 12 : */

-0.500000, -0.373016, 0.341270, 0.420635, 0.436508, 0.373016, 0.341270, 0.500000, 0.452381, 0.436508, 0.452381, 0.484127,
0.500000,

/* Output vector 12 : */

0.500000, -0.500000, -0.500000,

/* Input vector 13 : */

-0.500000, 0.418033, 0.500000, 0.336066, 0.327869, 0.336066, 0.336066, 0.360656, 0.360656, 0.352459, -0.155738, -0.049180,
-0.098361,

/* Output vector 13 : */

-0.500000, -0.500000, 0.500000,

/* Input vector 14 : */

-0.500000, 0.500000, 0.232143, 0.410714, 0.401786, 0.410714, 0.437500, 0.437500, 0.428571, 0.410714, -0.276786, -0.241071,
-0.258929,

/* Output vector 14 : */

-0.500000, -0.500000, 0.500000,

/* Input vector 15 : */

-0.465517, 0.500000, 0.241379, 0.413793, 0.405172, 0.413793, 0.413793, 0.439655, 0.439655, 0.431035, -0.103448, -0.500000,
-0.474138,

/* Output vector 15 : */

-0.500000, -0.500000, 0.500000,

/* Input vector 16 : */

-0.439850, 0.500000, 0.236842, 0.146617, 0.011278, -0.500000, 0.394737, 0.454887, 0.402256, 0.056391, -0.439850, 0.387218,
0.139098,

/* Output vector 16 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 17 : */

-0.418033, 0.278688, -0.500000, -0.106557, -0.254098, -0.073771, 0.163934, 0.229508, 0.500000, -0.204918, -0.418033, 0.155738,
-0.032787,

/* Output vector 17 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 18 : */

-0.433333, 0.275000, 0.316667, 0.216667, 0.066667, -0.500000, 0.491667, 0.500000, 0.500000, 0.116667, -0.433333, 0.483333, 0.208333,

/* Output vector 18 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 19 : */

-0.500000, 0.437008, 0.295276, 0.295276, 0.311024, 0.287402, 0.500000, 0.460630, 0.444882, 0.468504, 0.271654, 0.295276, 0.271654,

/* Output vector 19 : */

0.500000, -0.500000, -0.500000,

/* Input vector 20 : */

-0.491453, 0.286325, -0.380342, 0.465812, 0.491453, 0.294872, -0.500000, 0.072650, 0.294872, 0.405983, -0.405983, 0.500000, -0.405983,

/* Output vector 20 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 21 : */

-0.500000, 0.269231, 0.320513, 0.269231, 0.397436, 0.371795, 0.423077, 0.474359, 0.474359, 0.500000, 0.423077, 0.448718, 0.448718,

/* Output vector 21 : */

0.500000, -0.500000, -0.500000,

/* Input vector 22 : */

-0.500000, 0.214286, 0.392857, 0.357143, 0.428571, 0.464286, 0.392857, 0.428571, 0.392857, 0.357143, 0.428571, 0.464286, 0.500000,

/* Output vector 22 : */

0.500000, -0.500000, -0.500000,

/* Input vector 23 : */

-0.500000, 0.500000, 0.273723, 0.427007, 0.434307, 0.434307, 0.427007, 0.441606, 0.448905, 0.419708, 0.427007, 0.448905, 0.463504,

/* Output vector 23 : */

0.500000, -0.500000, -0.500000,

/* Input vector 24 : */

-0.500000, 0.269784, 0.435252, 0.428058, 0.442446, 0.420863, 0.399280, 0.435252, 0.420863, 0.442446, 0.478417, 0.500000, 0.478417,

/* Output vector 24 : */

0.500000, -0.500000, -0.500000,

/* Input vector 25 : */

-0.500000, -0.318841, 0.297101, 0.500000, -0.065217, 0.094203, -0.427536, 0.362319, -0.123188, -0.007246, -0.405797, 0.326087, 0.181159,

/* Output vector 25 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 26 : */

-0.500000, -0.345161, -0.035484, 0.435484, 0.441936, 0.448387, 0.454839, 0.461290, 0.461290, 0.435484, 0.448387, 0.480645,
0.500000,

/* Output vector 26 : */

0.500000, -0.500000, -0.500000,

/* Input vector 27 : */

-0.500000, 0.500000, -0.012821, -0.064103, -0.012821, 0.038462, 0.038462, -0.064103, -0.064103, -0.038462, -0.038462, -0.038462,
-0.038462,

/* Output vector 27 : */

0.500000, -0.500000, -0.500000,

/* Input vector 28 : */

-0.500000, 0.357143, 0.500000, 0.385714, 0.500000, 0.385714, 0.300000, 0.414286, 0.414286, 0.500000, 0.442857, 0.442857,
0.442857,

/* Output vector 28 : */

0.500000, -0.500000, -0.500000,

/* Input vector 29 : */

-0.500000, 0.407217, 0.448454, 0.448454, 0.489691, 0.500000, 0.500000, 0.469072, 0.479381, 0.479381, 0.479381, -0.293814,
-0.324742,

/* Output vector 29 : */

-0.500000, -0.500000, 0.500000,

/* Input vector 30 : */

-0.500000, -0.435484, 0.467742, 0.419355, 0.387097, 0.419355, 0.500000, 0.451613, 0.451613, 0.435484, 0.467742, 0.467742,
0.435484,

/* Output vector 30 : */

0.500000, -0.500000, -0.500000,

/* Input vector 31 : */

-0.500000, -0.079787, -0.340426, -0.250000, 0.260638, 0.500000, 0.250000, -0.297872, 0.159574, 0.271277, -0.271277, 0.250000,
0.239362,

/* Output vector 31 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 32 : */

-0.500000, -0.293103, 0.465517, 0.431035, 0.379310, 0.362069, 0.310345, 0.500000, 0.482759, 0.396552, 0.448276, 0.482759,
0.448276,

/* Output vector 32 : */

0.500000, -0.500000, -0.500000,

/* Input vector 33 : */

-0.500000, 0.088235, 0.205882, 0.500000, 0.360294, 0.330882, 0.367647, 0.323529, 0.382353, 0.375000, -0.235294, -0.227941,
-0.257353,

/* Output vector 33 : */

-0.500000, -0.500000, 0.500000,

/* Input vector 34 : */

-0.500000, 0.251880, 0.327068, 0.477444, 0.469925, 0.477444, 0.500000, 0.500000, 0.492481, 0.477444, -0.402256, -0.394737, -0.417293,

/* Output vector 34 : */

-0.500000, -0.500000, 0.500000,

/* Input vector 35 : */

-0.500000, 0.034483, -0.310345, -0.137931, -0.146552, -0.137931, -0.137931, -0.112069, -0.112069, -0.120690, 0.465517, 0.500000, 0.482759,

/* Output vector 35 : */

-0.500000, -0.500000, 0.500000,

/* Input vector 36 : */

-0.500000, 0.036232, 0.224638, -0.210145, -0.072464, -0.333333, 0.449275, -0.362319, 0.500000, 0.485507, -0.282609, -0.500000, 0.057971,

/* Output vector 36 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 37 : */

-0.343972, 0.173759, -0.500000, -0.251773, -0.223404, 0.053191, -0.003546, -0.088652, 0.500000, 0.457447, -0.209220, -0.372340, 0.109929,

/* Output vector 37 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 38 : */

-0.500000, -0.273109, 0.247899, 0.147059, 0.130252, 0.264706, 0.441176, 0.163866, 0.449580, 0.138655, -0.457983, 0.138655, 0.500000,

/* Output vector 38 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 39 : */

-0.214286, 0.500000, -0.238095, -0.261905, -0.309524, -0.428571, -0.500000, -0.380952, -0.428571, -0.404762, -0.238095, -0.190476, -0.285714,

/* Output vector 39 : */

0.500000, -0.500000, -0.500000,

/* Input vector 40 : */

-0.366667, 0.166667, -0.475000, 0.358333, -0.041667, 0.275000, -0.500000, -0.033333, 0.133333, 0.325000, 0.500000, -0.116667, 0.116667,

/* Output vector 40 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 41 : */

-0.500000, -0.388889, 0.293651, 0.404762, 0.357143, 0.436508, 0.468254, 0.420635, 0.436508, 0.420635, 0.500000, 0.468254, 0.436508,

/* Output vector 41 : */

0.500000, -0.500000, -0.500000,

/* Input vector 42 : */

-0.500000, -0.454000, -0.080000, 0.486667, 0.493333, 0.500000, 0.500000, 0.440000, 0.486667, 0.493333, 0.440000, 0.446667,
0.453333,

/* Output vector 42 : */

0.500000, -0.500000, -0.500000,

/* Input vector 43 : */

-0.500000, -0.005814, 0.343023, 0.284884, 0.500000, 0.459302, 0.343023, 0.360465, 0.360465, 0.354651, -0.354651, -0.337209,
-0.319767,

/* Output vector 43 : */

-0.500000, -0.500000, 0.500000,

/* Input vector 44 : */

-0.500000, -0.413386, 0.240157, 0.468504, 0.460630, 0.468504, 0.476378, 0.500000, 0.460630, 0.460630, 0.098425, 0.114173,
0.106299,

/* Output vector 44 : */

-0.500000, -0.500000, 0.500000,

/* Input vector 45 : */

-0.500000, 0.218310, 0.500000, 0.126761, 0.119718, 0.112676, 0.112676, 0.112676, 0.105634, 0.126761, 0.084507, -0.345070,
-0.359155,

/* Output vector 45 : */

-0.500000, -0.500000, 0.500000,

/* Input vector 46 : */

0.025862, 0.353448, 0.284483, 0.353448, -0.500000, -0.232759, -0.051724, 0.500000, 0.379310, 0.051724, -0.405172, -0.491379,
0.301724,

/* Output vector 46 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 47 : */

-0.455357, -0.500000, 0.437500, 0.383929, -0.062500, 0.187500, 0.437500, 0.107143, -0.080357, 0.428572, -0.401786, 0.133929,
0.500000,

/* Output vector 47 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 48 : */

-0.500000, 0.132911, 0.075949, 0.303797, 0.151899, 0.500000, -0.234177, 0.398734, -0.050633, -0.253165, 0.303797, 0.069620,
0.449367,

/* Output vector 48 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 49 : */

-0.500000, -0.089744, 0.493590, 0.493590, 0.500000, 0.500000, 0.461538, 0.493590, 0.480769, 0.487179, 0.474359, 0.487179,

0.493590,

/* Output vector 49 : */

0.500000, -0.500000, -0.500000,

/* Input vector 50 : */

-0.500000, 0.248252, -0.248252, 0.052448, 0.353147, 0.192308, -0.325175, -0.087413, 0.500000, -0.164336, 0.409091, 0.500000,
0.003497,

/* Output vector 50 : */

-0.500000, 0.500000, -0.500000,

/* Input vector 51 : */

-0.500000, -0.367647, 0.485294, 0.455882, 0.455882, 0.455882, 0.470588, 0.441176, 0.470588, 0.500000, 0.455882, 0.500000,
0.500000,

/* Output vector 51 : */

0.500000, -0.500000, -0.500000,

/* Input vector 52 : */

-0.500000, 0.500000, 0.354839, 0.338710, 0.451613, 0.435484, 0.435484, 0.435484, 0.435484, 0.403226, 0.435484, 0.435484,
0.322581,

/* Output vector 52 : */

0.500000, -0.500000, -0.500000,

/* Input vector 53 : */

-0.500000, 0.500000, 0.061224, 0.030612, -0.000000, 0.020408, -0.000000, 0.010204, -0.000000, -0.000000, -0.010204, -0.000000,
0.040816,

/* Output vector 53 : */

0.500000, -0.500000, -0.500000,

/* Input vector 54 : */

-0.500000, 0.500000, -0.253846, -0.284615, -0.284615, -0.346154, -0.269231, -0.269231, -0.361538, -0.253846, -0.253846, -0.269231,
-0.253846,

/* Output vector 54 : */

0.500000, -0.500000, -0.500000,

/* Input vector 55 : */

-0.282609, -0.021739, -0.500000, 0.091304, 0.500000, 0.039130, 0.282609, -0.482609, -0.013043, -0.421739, 0.413043, 0.152174,
-0.117391,

/* Output vector 55 : */

-0.500000, 0.500000, -0.500000,

END

SECTION B – SAMPLE DATA USED IN CHAPTER 9

Section B1 'Binary sequence'

Data for Chapter 9

```
ans =


  Columns 1 through 12

    0     0     0     0     1     1     1     1     0     0     0     0

Columns 13 through 24

    0     0     0     0     0     0     0     0     0     0     0     0

Columns 25 through 36

    0     0     0     0     0     0     0     0     1     1     1     1

Columns 37 through 48

    0     0     0     0     0     0     0     0     0     0     0     0

Columns 49 through 60

    0     0     0     0     1     1     1     1     0     0     0     0

Columns 61 through 72

    1     1     1     1     0     0     0     0     0     0     0     0

Columns 73 through 84

    0     0     0     0     1     1     1     1     0     0     0     0

Columns 85 through 96

    0     0     0     0     1     1     1     1     0     0     0     0
```

```
Columns 97 through 108

     0     0     0     0     0     0     0     0     1     1     1     1

Columns 109 through 120

     1     1     1     1     0     0     0     0     1     1     1     1

Columns 121 through 132

     0     0     0     0     0     0     0     0     1     1     1     1

Columns 133 through 144

     0     0     0     0     0     0     0     0     0     0     0     0

Columns 145 through 156

     1     1     1     1     0     0     0     0     0     0     0     0

Columns 157 through 168

     1     1     1     1     0     0     0     0     1     1     1     1

Columns 169 through 180

     0     0     0     0     1     1     1     1     0     0     0     0

Columns 181 through 192

     0     0     0     0     1     1     1     1     1     1     1     1

Columns 193 through 204

     0     0     0     0     0     0     0     0     1     1     1     1

Columns 205 through 216
```

```
    0     0     0     0     0     0     0     0     1     1     1     1

Columns 217 through 228

    1     1     1     1     1     1     1     1     0     0     0     0

Columns 229 through 240

    1     1     1     1     0     0     0     0     1     1     1     1
```

Section B2 'Noise –normal distribution –mean 0 variance 1'

```
noise =

Columns 1 through 7

 -1.2266   -0.1897   -0.3017    0.9570   -0.5334   -0.9011   -0.8926

Columns 8 through 14

  0.2787   -0.7458    1.6035    0.5743    0.3207   -0.1514    0.3158

Columns 15 through 21

  1.3437   -2.2378    1.2929   -0.3785    0.0025    0.8846    0.5825

Columns 22 through 28

 -1.6142   -1.5037    0.5736   -0.9105   -1.6313   -0.3591   -0.3976

Columns 29 through 35

 -1.1613   -1.1098    0.2907   -1.9102    1.3148    0.6653   -0.2751
```

Columns 36 through 42

-0.0230 -0.9080 -1.0437 0.3735 0.9015 1.2785 -0.1285

Columns 43 through 49

0.6128 1.9565 2.2663 -0.3740 2.2380 -0.1596 -0.7033

Columns 50 through 56

0.5635 -0.0503 1.1636 0.6588 -1.5501 -3.0291 0.5406

Columns 57 through 63

-1.0090 0.9080 1.5823 -0.9791 1.0079 0.1585 -0.5869

Columns 64 through 70

1.5741 -0.5166 1.2278 1.5839 -2.0890 2.9495 1.3561

Columns 71 through 77

1.0501 -0.7672 -0.2577 -1.3718 -1.2677 -0.8949 0.5891

Columns 78 through 84

1.8426 1.3480 -0.4913 -2.1776 0.2370 -0.7354 -1.7794

Columns 85 through 91

0.4480 0.5812 0.8566 -0.2663 -0.4175 -0.2058 -0.1743

Columns 92 through 98

0.2176 1.6843 0.1195 0.6507 2.0801 -0.3392 0.7301

Columns 99 through 105

0.2940 -0.8491 -2.5339 -2.3789 -0.3463 -0.6109 -0.4082

Columns 106 through 112

-1.4156 0.2271 0.2080 -0.7197 0.7578 -1.6431 -1.0568

Columns 113 through 119

-0.2517 -1.2984 1.2333 1.4946 0.2359 -1.4044 0.6588

Columns 120 through 126

-2.5566 -0.5349 3.2025 0.4392 -1.1499 0.8868 -0.2834

Columns 127 through 133

1.0353 -0.3649 1.3420 1.0089 0.2139 -0.2993 0.2558

Columns 134 through 140

-0.1908 -0.0791 0.6999 -0.7965 -0.8013 -0.0076 -0.7268

Columns 141 through 147

-1.4909 0.8703 -0.2657 -1.5667 -0.3946 -0.1439 -2.3342

Columns 148 through 154

-1.3575 -1.8157 1.1084 -0.1421 1.1128 0.5593 0.4784

Columns 155 through 161

-0.6794 0.2850 -1.3329 -0.7240 -0.6636 0.1984 -1.7949

Columns 162 through 168

-1.3877 0.1978 1.4693 0.3665 -0.4428 -0.0486 0.0777

Columns 169 through 175

1.9579 -0.0728 0.9388 -0.0796 -0.8010 0.3094 1.0518

Columns 176 through 182

-1.6642 -1.0908 -0.1917 0.4634 -0.9241 -0.6497 0.6229

Columns 183 through 189

-1.3351 1.0477 0.8633 -0.6424 0.6600 1.2941 0.3146

Columns 190 through 196

0.8596 0.1287 0.0166 -0.0728 -0.9943 -0.7474 -0.0308

Columns 197 through 203

0.9884 -0.5990 1.4766 -0.8138 0.6450 -1.3099 -0.8674

Columns 204 through 210

-0.4742 0.2224 1.8713 0.1100 -0.4113 0.5112 -1.1991

Columns 211 through 217

-0.0964 0.4458 -0.2958 -0.1680 0.1795 0.4211 1.6777

Columns 218 through 224

1.9969 0.6970 -1.3664 0.3630 -0.5670 -1.0442 0.6971

Columns 225 through 231

0.4840 -0.1938 -0.3781 -0.8864 -1.8402 -1.6282 -1.1738

Columns 232 through 238

-0.4154 0.1751 0.2294 -1.2409 0.7000 0.4269 1.4548

Columns 239 through 240

-0.5102 -0.0067

Section B3 'Binary sequence (B1) convoluted with noise (B2) '

```
»   [dnoise]'

Columns 1 through 7

   -1.2266   -0.1897   -0.3017    0.9570    0.4666    0.0989    0.1074

  Columns 8 through 14

    1.2787   -0.7458    1.6035    0.5743    0.3207   -0.1514    0.3158

 Columns 15 through 21

    1.3437   -2.2378    1.2929   -0.3785    0.0025    0.8846    0.5825

 Columns 22 through 28

   -1.6142   -1.5037    0.5736   -0.9105   -1.6313   -0.3591   -0.3976

 Columns 29 through 35

   -1.1613   -1.1098    0.2907   -1.9102    2.3148    1.6653    0.7249

 Columns 36 through 42

    0.9770   -0.9080   -1.0437    0.3735    0.9015    1.2785   -0.1285

 Columns 43 through 49
```

0.6128 1.9565 2.2663 -0.3740 2.2380 -0.1596 -0.7033

Columns 50 through 56

0.5635 -0.0503 1.1636 1.6588 -0.5501 -2.0291 1.5406

Columns 57 through 63

-1.0090 0.9080 1.5823 -0.9791 2.0079 1.1585 0.4131

Columns 64 through 70

2.5741 -0.5166 1.2278 1.5839 -2.0890 2.9495 1.3561

Columns 71 through 77

1.0501 -0.7672 -0.2577 -1.3718 -1.2677 -0.8949 1.5891

Columns 78 through 84

2.8426 2.3480 0.5087 -2.1776 0.2370 -0.7354 -1.7794

Columns 85 through 91

0.4480 0.5812 0.8566 -0.2663 0.5825 0.7942 0.8257

Columns 92 through 98

1.2176 1.6843 0.1195 0.6507 2.0801 -0.3392 0.7301

Columns 99 through 105

0.2940 -0.8491 -2.5339 -2.3789 -0.3463 -0.6109 0.5918

Columns 106 through 112

-0.4156 1.2271 1.2080 0.2803 1.7578 -0.6431 -0.0568

Columns 113 through 119

-0.2517 -1.2984 1.2333 1.4946 1.2359 -0.4044 1.6588

Columns 120 through 126

-1.5566 -0.5349 3.2025 0.4392 -1.1499 0.8868 -0.2834

Columns 127 through 133

1.0353 -0.3649 2.3420 2.0089 1.2139 0.7007 0.2558

Columns 134 through 140

-0.1908 -0.0791 0.6999 -0.7965 -0.8013 -0.0076 -0.7268

Columns 141 through 147

-1.4909 0.8703 -0.2657 -1.5667 0.6054 0.8561 -1.3342

Columns 148 through 154

-0.3575 -1.8157 1.1084 -0.1421 1.1128 0.5593 0.4784

Columns 155 through 161

-0.6794 0.2850 -0.3329 0.2760 0.3364 1.1984 -1.7949

Columns 162 through 168

-1.3877 0.1978 1.4693 1.3665 0.5572 0.9514 1.0777

Columns 169 through 175

1.9579 -0.0728 0.9388 -0.0796 0.1990 1.3094 2.0518

Columns 176 through 182

-0.6642 -1.0908 -0.1917 0.4634 -0.9241 -0.6497 0.6229

Columns 183 through 189

-1.3351 1.0477 1.8633 0.3576 1.6600 2.2941 1.3146

Columns 190 through 196

1.8596 1.1287 1.0166 -0.0728 -0.9943 -0.7474 -0.0308

Columns 197 through 203

0.9884 -0.5990 1.4766 -0.8138 1.6450 -0.3099 0.1326

Columns 204 through 210

0.5258 0.2224 1.8713 0.1100 -0.4113 0.5112 -1.1991

Columns 211 through 217

-0.0964 0.4458 0.7042 0.8320 1.1795 1.4211 2.6777

Columns 218 through 224

2.9969 1.6970 -0.3664 1.3630 0.4330 -0.0442 1.6971

Columns 225 through 231

0.4840 -0.1938 -0.3781 -0.8864 -0.8402 -0.6282 -0.1738

Columns 232 through 238

0.5846 0.1751 0.2294 -1.2409 0.7000 1.4269 2.4548

Columns 239 through 240

0.4898 0.9933

Section B4 'Binary sequence convoluted with 20% noise (B2)'

```
d20noise =


  Columns 1 through 7

   -0.2453   -0.0379   -0.0603    0.1914    0.8933    0.8198    0.8215

  Columns 8 through 14

    1.0557   -0.1492    0.3207    0.1149    0.0641   -0.0303    0.0632

  Columns 15 through 21

    0.2687   -0.4476    0.2586   -0.0757    0.0005    0.1769    0.1165

  Columns 22 through 28

   -0.3228   -0.3007    0.1147   -0.1821   -0.3263   -0.0718   -0.0795

  Columns 29 through 35

   -0.2323   -0.2220    0.0581   -0.3820    1.2630    1.1331    0.9450

  Columns 36 through 42

    0.9954   -0.1816   -0.2087    0.0747    0.1803    0.2557   -0.0257

  Columns 43 through 49

    0.1226    0.3913    0.4533   -0.0748    0.4476   -0.0319   -0.1407

  Columns 50 through 56

    0.1127   -0.0101    0.2327    1.1318    0.6900    0.3942    1.1081
```

Columns 57 through 63

-0.2018 0.1816 0.3165 -0.1958 1.2016 1.0317 0.8826

Columns 64 through 70

1.3148 -0.1033 0.2456 0.3168 -0.4178 0.5899 0.2712

Columns 71 through 77

0.2100 -0.1534 -0.0515 -0.2744 -0.2535 -0.1790 1.1178

Columns 78 through 84

1.3685 1.2696 0.9017 -0.4355 0.0474 -0.1471 -0.3559

Columns 85 through 91

0.0896 0.1162 0.1713 -0.0533 0.9165 0.9588 0.9651

Columns 92 through 98

1.0435 0.3369 0.0239 0.1301 0.4160 -0.0678 0.1460

Columns 99 through 105

0.0588 -0.1698 -0.5068 -0.4758 -0.0693 -0.1222 0.9184

Columns 106 through 112

0.7169 1.0454 1.0416 0.8561 1.1516 0.6714 0.7886

Columns 113 through 119

-0.0503 -0.2597 0.2467 0.2989 1.0472 0.7191 1.1318

Columns 120 through 126

0.4887 -0.1070 0.6405 0.0878 -0.2300 0.1774 -0.0567

Columns 127 through 133

0.2071 -0.0730 1.2684 1.2018 1.0428 0.9401 0.0512

Columns 134 through 140

-0.0382 -0.0158 0.1400 -0.1593 -0.1603 -0.0015 -0.1454

Columns 141 through 147

-0.2982 0.1741 -0.0531 -0.3133 0.9211 0.9712 0.5332

Columns 148 through 154

0.7285 -0.3631 0.2217 -0.0284 0.2226 0.1119 0.0957

Columns 155 through 161

-0.1359 0.0570 0.7334 0.8552 0.8673 1.0397 -0.3590

Columns 162 through 168

-0.2775 0.0396 0.2939 1.0733 0.9114 0.9903 1.0155

Columns 169 through 175

0.3916 -0.0146 0.1878 -0.0159 0.8398 1.0619 1.2104

Columns 176 through 182

0.6672 -0.2182 -0.0383 0.0927 -0.1848 -0.1299 0.1246

Columns 183 through 189

-0.2670 0.2095 1.1727 0.8715 1.1320 1.2588 1.0629

```
Columns 190 through 196

  1.1719    1.0257    1.0033   -0.0146   -0.1989   -0.1495   -0.0062

Columns 197 through 203

  0.1977   -0.1198    0.2953   -0.1628    1.1290    0.7380    0.8265

Columns 204 through 210

  0.9052    0.0445    0.3743    0.0220   -0.0823    0.1022   -0.2398

Columns 211 through 217

 -0.0193    0.0892    0.9408    0.9664    1.0359    1.0842    1.3355

Columns 218 through 224

  1.3994    1.1394    0.7267    1.0726    0.8866    0.7912    1.1394

Columns 225 through 231

  0.0968   -0.0388   -0.0756   -0.1773    0.6320    0.6744    0.7652

Columns 232 through 238

  0.9169    0.0350    0.0459   -0.2482    0.1400    1.0854    1.2910

Columns 239 through 240

  0.8980    0.9987
```

Section B5 'Pink noise'

```
pink =

  Columns 1 through 7

   -0.1507   -0.1705   -0.1370   -0.0022   -0.0230   -0.1956   -0.2489

  Columns 8 through 14

   -0.1627   -0.1940   -0.0219    0.1566    0.0305    0.0853   -0.0280

  Columns 15 through 21

    0.2957   -0.1569    0.0371    0.0628   -0.0189    0.1299    0.2009

  Columns 22 through 28

   -0.0447   -0.2781   -0.0941   -0.1107   -0.3633   -0.2861   -0.2655

  Columns 29 through 35

   -0.3211   -0.4691   -0.2553   -0.4894   -0.2710   -0.0536   -0.1737

  Columns 36 through 42

   -0.1568   -0.2504   -0.3838   -0.2553   -0.1022    0.0753   -0.0055

  Columns 43 through 49

    0.0095    0.2265    0.4981    0.2677    0.4315    0.4230    0.1099

  Columns 50 through 56

    0.2522    0.1766    0.3212    0.3663    0.1450   -0.3375   -0.1271

  Columns 57 through 63
```

-0.1130 -0.0588 0.2974 0.0380 0.1394 0.2042 0.0296

Columns 64 through 70

0.2491 0.2046 0.1918 0.5445 0.0339 0.4168 0.6190

Columns 71 through 77

0.5702 0.3959 0.2646 0.1423 -0.0501 -0.0536 0.0425

Columns 78 through 84

0.3644 0.4682 0.3566 -0.0437 0.0400 0.0715 -0.2005

Columns 85 through 91

-0.0490 0.1053 0.2046 0.1592 0.0525 0.0585 0.0474

Columns 92 through 98

0.0795 0.3055 0.3169 0.2462 0.5601 0.3961 0.3825

Columns 99 through 105

0.4254 0.2483 -0.0609 -0.3490 -0.2029 -0.1650 -0.1381

Columns 106 through 112

-0.2623 -0.1900 -0.0274 -0.1616 -0.0018 -0.1609 -0.3468

Columns 113 through 119

-0.1962 -0.3629 -0.1201 0.1255 0.1394 -0.1556 -0.0162

Columns 120 through 126

-0.2761 -0.4202 0.1986 0.2597 -0.0729 0.0753 0.0440

Columns 127 through 133

0.1419 0.1061 0.1818 0.3670 0.2522 0.1996 0.1575

Columns 134 through 140

0.1899 0.0856 0.2534 0.0928 -0.0038 0.0055 0.0161

Columns 141 through 147

-0.2358 -0.0048 0.0175 -0.1866 -0.2355 -0.0785 -0.4094

Columns 148 through 154

-0.4512 -0.6001 -0.2930 -0.2222 -0.0951 0.0050 0.0210

Columns 155 through 161

-0.0961 -0.0667 -0.1918 -0.3001 -0.2795 -0.2113 -0.3445

Columns 162 through 168

-0.5499 -0.3542 -0.1250 -0.0093 -0.1913 -0.1262 -0.1821

Columns 169 through 175

0.1518 0.0676 0.1164 0.0998 -0.0686 -0.0444 0.1607

Columns 176 through 182

-0.0942 -0.2725 -0.1942 -0.0817 -0.1504 -0.2837 -0.0765

Columns 183 through 189

-0.2550 -0.1300 0.0920 -0.0900 -0.0187 0.1678 0.1568

Columns 190 through 196

0.1821 0.1950 0.0937 0.1237 -0.0652 -0.1034 -0.1196

Columns 197 through 203

0.1268 -0.0346 0.1847 0.0567 0.0831 -0.0243 -0.2050

Columns 204 through 210

-0.1351 -0.1221 0.1988 0.1832 0.0225 0.1130 -0.0430

Columns 211 through 217

-0.0932 0.0602 -0.0025 -0.0219 0.0123 0.0677 0.2550

Columns 218 through 224

0.4783 0.4690 0.1587 0.1497 0.1652 -0.0591 0.1141

Columns 225 through 231

0.1883 0.1521 0.0605 0.0006 -0.2374 -0.3402 -0.4013

Columns 232 through 238

-0.3031 -0.2325 -0.0848 -0.2949 -0.1335 -0.0558 0.1430

Columns 239 through 240

0.323 -0.0162

Section B6 'Binary sequence' convoluted with pink noise'

```
dpink =

  Columns 1 through 7

   -0.1507   -0.1705   -0.1370   -0.0022    0.9770    0.8044    0.7511

  Columns 8 through 14

    0.8373   -0.1940   -0.0219    0.1566    0.0305    0.0853   -0.0280

Columns 15 through 21

    0.2957   -0.1569    0.0371    0.0628   -0.0189    0.1299    0.2009

Columns 22 through 28

   -0.0447   -0.2781   -0.0941   -0.1107   -0.3633   -0.2861   -0.2655

Columns 29 through 35

   -0.3211   -0.4691   -0.2553   -0.4894    0.7290    0.9464    0.8263

Columns 36 through 42

    0.8432   -0.2504   -0.3838   -0.2553   -0.1022    0.0753   -0.0055

Columns 43 through 49

    0.0095    0.2265    0.4981    0.2677    0.4315    0.4230    0.1099

Columns 50 through 56

    0.2522    0.1766    0.3212    1.3663    1.1450    0.6625    0.8729

Columns 57 through 63

   -0.1130   -0.0588    0.2974    0.0380    1.1394    1.2042    1.0296
```

Columns 64 through 70

1.2491 0.2046 0.1918 0.5445 0.0339 0.4168 0.6190

Columns 71 through 77

0.5702 0.3959 0.2646 0.1423 -0.0501 -0.0536 1.0425

Columns 78 through 84

1.3644 1.4682 1.3566 -0.0437 0.0400 0.0715 -0.2005

Columns 85 through 91

-0.0490 0.1053 0.2046 0.1592 1.0525 1.0585 1.0474

Columns 92 through 98

1.0795 0.3055 0.3169 0.2462 0.5601 0.3961 0.3825

Columns 99 through 105

0.4254 0.2483 -0.0609 -0.3490 -0.2029 -0.1650 0.8619

Columns 106 through 112

0.7377 0.8100 0.9726 0.8384 0.9982 0.8391 0.6532

Columns 113 through 119

-0.1962 -0.3629 -0.1201 0.1255 1.1394 0.8444 0.9838

Columns 120 through 126

0.7239 -0.4202 0.1986 0.2597 -0.0729 0.0753 0.0440

Columns 127 through 133

0.1419 0.1061 1.1818 1.3670 1.2522 1.1996 0.1575

Columns 134 through 140

0.1899 0.0856 0.2534 0.0928 -0.0038 0.0055 0.0161

Columns 141 through 147

-0.2358 -0.0048 0.0175 -0.1866 0.7645 0.9215 0.5906

Columns 148 through 154

0.5488 -0.6001 -0.2930 -0.2222 -0.0951 0.0050 0.0210

Columns 155 through 161

-0.0961 -0.0667 0.8082 0.6999 0.7205 0.7887 -0.3445

Columns 162 through 168

-0.5499 -0.3542 -0.1250 0.9907 0.8087 0.8738 0.8179

Columns 169 through 175

0.1518 0.0676 0.1164 0.0998 0.9314 0.9556 1.1607

Columns 176 through 182

0.9058 -0.2725 -0.1942 -0.0817 -0.1504 -0.2837 -0.0765

Columns 183 through 189

-0.2550 -0.1300 1.0920 0.9100 0.9813 1.1678 1.1568

Columns 190 through 196

1.1821 1.1950 1.0937 0.1237 -0.0652 -0.1034 -0.1196

Columns 197 through 203

0.1268 -0.0346 0.1847 0.0567 1.0831 0.9757 0.7950

Columns 204 through 210

0.8649 -0.1221 0.1988 0.1832 0.0225 0.1130 -0.0430

Columns 211 through 217

-0.0932 0.0602 0.9975 0.9781 1.0123 1.0677 1.2550

Columns 218 through 224

1.4783 1.4690 1.1587 1.1497 1.1652 0.9409 1.1141

Columns 225 through 231

0.1883 0.1521 0.0605 0.0006 0.7626 0.6598 0.5987

Columns 232 through 238

0.6969 -0.2325 -0.0848 -0.2949 -0.1335 0.9442 1.1430

Columns 239 through 240

1.0323 0.9838

APPENDIX B - SOFT COMPUTING: DEVELOPMENT AND METHODOLOGIES

As a consultant at Fisher Controls International, Leicester, in 1995/1996, I carried out suitability studies of ANNs in process control and published two internal papers. The material included in this Appendix, which covers preliminary unifying definitions and introductory information on neuro-computing, form part of one of those publications.

Historical

An understanding of the human mind, its construction and where it is placed in the body was extensively sought by the philosophers of ancient civilisations. The arrangement of the blood vessels led the Sumerians and Asyrians to the conclusion that the liver and later the heart was the animator ["anima" Latin for soul] or mover of body.

A great deal of research in neurobiology and psychology has been carried out in this century. As far as computer scientists are concerned, the pioneering investigation in soft computing paradigms was started by Warren S. McCulloch and Walter Pitts [McCulloch and Pitts, 1943]. This influential study proposed an artificial neuron or processing element acting as a simple threshold logic device. They concluded that a network of these processing elements, connected to one another, may achieve significant computational power.

Since 1943, many researchers have investigated the fundamental concepts of such networks. In 1949, Donald Hebb's 'Organisation of Behaviour' [Hebb, 1961] had a great impact in the scientific community. In 1958, F.Rosenblatt proposed his learning machine the perceptron [Rosenblatt, 1958]. This lead to the first era of wide spread interest in neural computing. He also wrote 'Principles of Neurodynamics' [Rosenblatt,1961], one of the first books on neurocomputing. Of other major works in the early 1960's, Bernard Widrow's processing element, 'the adaline' which was equipped with a powerful learning law, is still in use. In 1965, L. A. Zadeh introduced the concept of fuzzy sets, but despite this, by the end of the 1960's neural computing's first era of success was drawn to a close. A campaign led by Marvin Minsky and Seymour Papert managed to divert almost all of the research funding in this field to the field of 'artificial intelligence' [Hecht-Nielsen, 1987]. In the publication Perceptrons [Minsky and Papert, 1969], the limitations of Perceptrons, a restricted class of neural systems, were discussed, implying that neurocomputing research is a proven dead end. Following this, throughout the 1970's, serious neural network researchers [e.g. Teuvo Kohonen, Stephen Grossberg, James A.Anderson and Leon N.Cooper] managed to attract research funds by describing their work under the headings of 'adaptive signal processing', 'pattern recognition', and 'biological modelling'.

Recent

In 1983, the American defence advanced research project agency [DARPA], began funding neuro-computing research. This action opened the floodgates [Hecht-Nielson, 1987]. During the 1980's greater attention was also paid to Generic Algorithms (G.A.), first developed in 1975 [Holland, 75], and other derivative free stochastic optimisation methods e.g. simulated annealing.

The following work, during the 1980's, re-stimulated research interests in neural networks and helped to bring about the modern era of spectacular developments:

The work of J.J. Hopfield [Hopfield, 82; Hopfield, 84] brought together a number of separate earlier neural network ideas.

The development of the back propagation algorithm by several investigators.

The publication of parallel distributed processing, edited by D.Rumelhart, J. McClelland [Rumelhart and McClelland, 1986]

In 1987 the American Institute of Electrical and Electronic Engineering held its first international conference on neural networks. The International Neural Network Society [INNS] was formed in 1988. Many neural network journals began publishing in the late 1980's and early 1990's; e.g. the INNS journal 'Neural Networks',1988; the IEEE Transaction on Neural Networks, 1990; Neural Computing and Applications, 1993. During the late 1990's, the concept of soft computing and their applications to real world problems has **rapidly become a reality.**

Unifying definitions and concepts

- **Learning and Training**

Learning is defined as a permanent change in behaviour. Training is the process by which leaning occurs. In artificial neural networks training is referred to a systematic method of modifying the weights in a network to improve the response of the network to a collection of training examples.

In a network of distributed processing elements, modification of the connection strength is performed by the processing elements using a leaning law. Learning laws [or rules] are the algorithms for varying the weights such that learning occurs. In order to understand learning law equations, the information environment where they operate, the weight modification process, and the training method should be considered.

- **Information environment**

In a distributed information processing structure each processing element responds separately and in isolation to the incoming signals of one or more input classes. The response is in accordance with a learning law equation.

To understand the learning process, the information environment in which the entire network operates should be envisaged. Information environments are usually described in terms of probability density functions. For most learning laws, constraints are placed on the information environment used.

- **Weight modification process**

In a network of n processing elements with adaptive weights, a network weight vector W can be formed by combining all the weights of all the processing elements. The idea is that if the information processing performance required is to be realised by this network, it will be found at some value of the vector W.

Learning laws should efficiently guide the weight vector W to a location that yields the desired network performance. Learning law objectives are application dependent. For example, in pattern classification the objective is to classify and predict successfully on new data. In control applications it can be to approximate non-linear functions and to make unknown systems follow the desired response.

Energy

The energy concept is an important tool for analysing and explaining some of the properties of neural models. The concept is best described by [Hopfield and Tank, 1986]. Energy is defined by an explicit mathematical formula that depends on the characteristics of the network [e.g. the interconnection strength]. The computational energy of a network can be pictured as a landscape of hills and valleys.

Using the energy concept, every possible pattern of firing and non firing of a network can be defined. With every change in the firing pattern, the energy reduces accordingly or stays the same. Eventually the last state is reached. This stable state represents the minimum energy state. A stable state is the equivalent of being at the bottom of some valley or energy well. Mathematically, a function called 'Liapunov function' works in a similar manner. Further explanation on the computational energy concept is provided in working paper number 44 [Osanlou, 1992b].

Dynamic and adaptive structures

There are numerous neural network based soft computing paradigms. Despite appearances they all perform the same function. They accept an input vector and produce a corresponding output vector, and may be viewed as vector mappers. Likewise all neural network applications may be viewed as special cases of vector mapping [Wasserman, 1993]. If the output vector is identical to the input vector, the network is said to be an auto-associative mapper. If the output vector produced is different from the input vector on which it was trained, the network is referred to as a hetero-associative mapper.

In the categorisation of the various soft computing paradigms, with the biological and artificial neuronal model as the basis, the mapping relationship is usually used to classify various structures. They may be either 'static', 'dynamic', or alternatively 'fuzzy systems' or unconventional approaches [e.g. 'CMAC' and 'neurone with hysteresis']

Differences: between neural networks and conventional computers

The main contributions of ANNs is in their ability to:

learn by example,

generalise to process [possibly incomplete] new data that the network has not seen before,

operate on noisy data that occurs in real world applications, and

approximate arbitrary non-linear functions, making them useful to model non-linear systems.

Neural Network	**Conventional Computers**
Structure. *Inherently parallel, leading to the possibility of fast hardware implementations.*	**Structure** *Linear design approach.*
Processing method. *Processing occurs as a distributed pattern in response to an input pattern, controlled by the learning law, processing element interconnections, and transfer functions.*	**Processing method**. *Processing occurs in the form of fetch, execute, and store.*
Programming method. *Networks are trained rather than programmed, i.e. they learn by example. The learning process is conducted by rules which often govern modifications of the weights. of the processing elements. Once trained, a network is capable of generalising when presented with new inputs not used in the training set.*	**Programming method**. *A programme of sequential instructions. Superior in well defined applications, with a large amount of numerical operations and accurately specified repeatable steps.*
Memory. *Memory and processing elements are the same. Data is stored as patterns represented by variable weights of*	**Memory**. *Memory and processing elements are separate.*

the network's processing elements.

Fault tolerant. *Also called "graceful degradation". Performance drops gradually in response to defects.* **Not fault tolerant**.

Robustness. *Networks can operate on noisy data that occurs in real world applications.* **No robustness to noisy or incomplete data**.

Configurations most applicable to pattern recognition

Artificial neural network architectures described so far, have all been used extensively in signal analysis and applications such as identification of objects from observed patterns or images. Of those architectures, the multilayer neural network with an error backpropagation algorithm is the most commonly used. Other common architectures in signal processing and pattern recognition which have not been mentioned yet, are briefly covered here.

- **Hopfield**

The Hopfield model belongs to a class of associative memory known as crossbar nets, because of their resemblance to the primitive operator controlled telephone systems in which every input and output telephone line could be connected through a single intersection switch. General architectural characteristics of such networks, which includes Bidirectional Associative Memory [BAM] and Brain State in a Box, are as follows: 1] all the processing elements are fully interconnected, i.e. any input and output is connected through a variable weight, 2] nets consist of either 'a single layer' or 'an input layer and an output layer'.

Each node in the input layer of a Hopfield model is directly connected to only one node in the Hopfield layer. The processing elements in the second layer may have hard limiter or sigmoidal activation functions. Their weighted outputs are fed back to the input of the rest of the processing elements.

Prior to operation, weights are adjusted in a way that the input and output of the network are the same. Then an input pattern is applied and the outputs are continually fed back through the weights until a convergence criterion is met [Page et al., 1993].

If in the convergence process a local minima is reached, Hopfield provides no way of reaching the global minima from there. Hinton and Sejnowski, provide a solution to this problem with the Boltzmann machine [Hinton and Sejnowski, 1986].

Kohonen

The Kohonen network has self organising properties and is capable of pattern categorisation. It is constructed from a fully interconnected array of processing elements. The output of each processing element is the input to all processing elements including itself. Each processing element receives the input pattern which consists of a set of continuous valued data. There are two sets of weights: an adaptive set, to compute the weighted sum of the external inputs, and a fixed set between neurones, that controls neurone interactions in the network.

Training involves applying an input pattern to the network and computing each processing element's output. After interacting with each other, only the processing element with the largest output and neighbouring processing elements are allowed to adjust their weights.

- **Counter propagation networks**

Counter-propagation networks are multi layer hybrid dynamic [feedback] structures. They perform continuous mappings by first feeding data into a Kohonen network using unsupervised training and then feeding the output to a Grossberg outstar with supervised training [Wasserman, 1989]. They are fast but lack the generality of backpropagation.

Radial basis-function networks (RBF)

Radial basis function structures, are feed forward multi layer networks. In a similar manner to all feed forward networks with a hidden layer of non-linear processing elements, they are universal approximators [Girosi et al., 1991 and Hartman et al., 1990]. They can approximate any continuous function with arbitrary accuracy. The non-linearity need not be sigmoidal [Stinchcome and White, 1989].

Training is usually orders of magnitude faster than backpropagation. None of backpropagation's training problems, previously discussed in publication number 44, [see Osanlou, 1992b], are experienced with RBF.

Their main disadvantages are requiring all, or a substantial portion, of the training set to be involved in their operation, and slow operation after completion of the rapid, reliable training. This is due to the large number of variables involved.

Similar and closely related networks are referred to as: 'Gaussian potential function' [Lee and Kil, 1991]; 'localised receptive fields' [Moody and Darken, 1988]; 'regularisation networks' [Poggio and Girosi, 1990]; and 'locally tuned processing units' [Moody and Darken, 1989].

APPENDIX C – PUBLICATION

Modified versions of the paper presented here, have appeared in:

- Recent Advances in Soft Computing'98

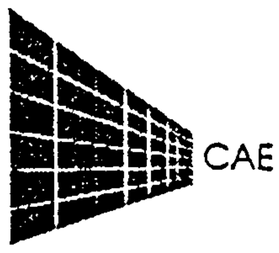
  July 2nd and 3rd 1998

  DMU, Leicester.

- Technical Monograph 27

  Science and Engineering Research Centre

  DMU, Leicester.



3rd International Scientific Colloquium

CAE TECHNIQUES

Proceedings of the 3rd International Scientific Colloquium
Rzeszów University of Technology, Poland
September 24-27, 1997

edited by
KAZIMIERZ E. OCZOŚ

Pattern recognition using artificial neural networks with white noise for process control engineering

Jonathan M. Blackledge, Ardishir Osanlou

De Montfort University, School of Computer Sciences, Department of Mathematical Sciences, The Gateway, Leicester LE1 9BH, UNITED KINGDOM

Abstract: *This paper discusses the most significant findings of a comprehensive study on the use and suitability of Artificial Neural Networks (ANNs) for process modelling and control. Using a pattern recognition problem associated with a fluid flow process control unit analogous to those used in the petrochemicals industries, this study has shown that care needs to be exercised in the choice of network structure and training and that attributes, such as robustness in the presence of noise may actually enable ANNs to improve process control engineering in many ways. By developing a robust weight set, their robustness to noisy data has been examined. It has been observed that adding white noise greatly improves the performance of the networks both in terms of training and application. In other words the added noise actually helps the networks in their pattern recognition task. Since noise is a major problem for real world applications, this is a particularly important result which is further analysed and discussed in this paper.*

Keywords: pattern recognition, Artificial Neural Networks, process control engineering, white noise.

1. Introduction

Artificial Neural Networks (ANNs) are often applied to complex problems in which conventional computer algorithms are not appropriate or difficult to design. They have many attributes and characteristics which make them fundamentally different from conventional Von Neumann type systems and are based on algorithms derived from biological models with the aim of producing similar characteristics, e.g. immunity to noise or adaptability to new circumstances.

Neural networks can be described as parallel distributed information processing structures made up of processing elements; the elements being simplified versions' of biological neurons. Because they are not based on rigid algorithms, ANNs are naturally fault tolerant. They can process many inputs and produce many outputs; hence, they are applicable to multivariable systems and potentially offer fast processing. Other major attributes of ANNs include their ability to learn by example, their generalising ability to process (possibly incomplete) new data that the network has not seen before and the ability to operate on noisy data that occur in real world applications.

Many different ANN paradigms are available, but basically they all perform the same function (i.e. they accept a set of inputs and produce a set of outputs) and have many features in common. Likewise all ANN applications are special cases of vector mappings. If the output vector is identical to the input vector, the network is said to be an auto-associative mapper. If the output vector produced is different from the input vector on which it was trained, the network is referred to as a hetero-associative mapper.

The mapping relationship is usually used to classify various neural structures. They may be either static or dynamic, or based on a variety of alternatives such as "fuzzy systems" and unconventional approaches such as the "Cerebellar Model Articulation Controller" (CMAC) and the "Neurone with Hysteresis" paradigm.

1.1. Static structures

Statistic structures have no feed-back and are inherently stable. The network response depends on its current inputs and the weight values. During the learning process, externally imposed feed-back interaction takes place. Once trained, each application of a given input set always produces the same output set. Static structures are also referred to as feed-forward, or non-recurrent systems. Examples include the "single layer'" and "multi layer" perceptron and "Radial Basis Function" networks.

1.2. Dynamic structures

Dynamic structures employ feed-back between the processing elements, which allows them to produce dynamic mappings. Referred to as recurrent networks, they have local memory characteristics. One of the advantages they offer over purely static networks is that their output depends upon previous, as well as current inputs and/or outputs. Examples of dynamic structures include the "Hopfield-Tank" model and the "Learning Vector Quantization" model (for single layer networks); Multi-Layer Feed-Forward Feed-back networks; cellular, time delay, and Counter Propagation models (for multi-layer hybrid structures); first order dynamics and second order dynamics (for multi-layer excitatory and inhibitory systems).

1.3. Learning and training

Learning is defined as a permanent change in behaviour. Training is the process by which leaning occurs. In ANNs, training is referred to as a systematic method of

modifying the weights (or the connection strength) in a network to improve the response of the network to a collection of training examples. In a network of distributed processing elements, modification of the weights is performed by the processing elements using a leaning law. Learning laws (or rules) are the algorithms for varying the weights such that learning occurs.

To understand the learning process, the information environment in which the entire network operates should be envisaged. Information environments are usually described in terms of probability density functions. For most learning laws, constraints are placed on the information environment used. In a network of n processing elements with adaptive weights, a network weight vector w can be formed by combining all the weights of all the processing elements. If the information processing performance required is to be realised by this network, it will be found at some value of the vector w. Learning laws should efficiently guide the weight vector w to a location that yields the desired network performance. Learning law objectives are application dependent. Learning algorithms may be broadly categorised as error based or output based.

Error based learning algorithms

In this scheme, an external reference signal is compared with the obtained response. The error signal is then used to modify the weights to improve the system performance. Error based (or supervised) learning can be divided into two important categories: error correction (e.g. least mean square and back-propagation) and stochastic [7]. Stochastic learning algorithms make random changes to the weights. The resultant " energy" created by the changes is then determined. The weight changes are retained if the "energy" is lowered and a pre-defined probability distribution function can be used to keep the weight change, even if the energy is not lowered.

Output based learning algorithms

If the desired (target) results are unknown, then error based learning algorithms cannot be used and output based learning algorithms become useful. Such algorithms do not employ any external reference signal. Self organisation principles and internal control mechanisms are generally used to classify the input data and to discover collective properties. Important examples of output based (or unsupervised) learning algorithms include hebbian learning and competitive learning.

1.4. Significant characteristic of ANNs

One of the most significant characteristics of ANNs is their ability to approximate arbitrary non-linear functions. This ability makes them useful in modelling non-linear systems and means that neural technology is of growing importance in industrial applications, e.g. process control systems with multiple inputs and outputs. For example, many of the limitations of conventional control solutions arise in trying to control poorly

modelled non-linear systems; unrealistically assuming that there is no noise in the measurements.

By utilising the Weierstrass theorem, it can be shown that a feed-forward network, with only one hidden layer, where each processing element in the hidden layer has a continuous sigmoidal non-linearity, can approximate a continuous function [1]. However, networks with two hidden layers tend to provide better generalisations and higher accuracy than networks with a single hidden layer [3]. Gallant and White [5] demonstrated that a feed-forward network with a single hidden layer using a cosine activation function, is capable of embedding a Fourier network. The network possesses all the approximation properties of a Fourier series representation.

Application of the Kolmogorov theorem has demonstrated the function approximation capabilities of feed-forward networks [6]. However, Poggio and Girosi [8] have pointed out that Kolmogorov's theorem requires a different non-linear processing function for each unit in the network, and that functions in the second hidden layer depend on the function being approximated. For multi-layer static neural networks, the most commonly used learning rule is a generalisation of the least squares rule, known as the Back-Propagation (BP) algorithm. Its main objective is to improve the information content encoded in the weights, by reducing the error at each node. The multi-layer neural network with a BP algorithm has been used extensively in many control applications, many of which encounter problems such as large error or slow adaptation. Algebraic and trigonometric polynomials, share the multi-layer feed-forward network property of approximating arbitrarily well a continuous function. However, from an approximation theory viewpoint of characterising good approximation schemes, the property of best approximation is more critical than the property of approximating continuous functions arbitrarily well. An approximation scheme is said to have the best approximation property if, in the set of approximating functions, there is one that has minimum distance from any given function of a larger set (a precise mathematical formulation is given by Poggio and Girosi [8]; see also Watson [10]). Poggio and Girosi [8] have shown that multi-layer perceptron networks, used with BP, do not have the best approximation property. For regularisation networks (in particular, Radial Basis Function networks) they show the existence and uniqueness of best approximation.

One of the most widely used static neural networks is the Radial Basis Function (RBF) network in which it can be shown [8] that networks derived from regularisation theory have a similar property to multi-layer perceptrons, i.e. they can approximate arbitrarily well continuous functions. CMAC is another popular static neural network. From a mathematical viewpoint, the CMAC learning rule, when applied to the CMAC functional form, is equivalent to BP. Both RBF and CMAC have a key advantage over multi-layer perceptrons in real-time learning because they adapt different sets of weights for different regions of the input-space. Least squares methods (e.g. BP), have greater potential in real time learning when applied to such networks. For example, using BP with networks similar to RBF for pattern recognition can increase learning rates [4].

2. A problem in process control engineering

The real worth of neural computing technology must be measured in its application to real world problems. Employing a benchmark neuro-control problem yields important considerations for developing neuro-control solutions (e.g. noise, non-linear processes and the incorporation of approximate models) and ensuring that optimum results are analysed experimentally.

In this section, a problem in process control engineering is described and the approaches to solving the problem using ANNs discussed. Details of the experiments and their evaluation are given in Section 3.

The objectives of this benchmark pattern recognition problem were as follows:

- to address control issues regarding noise and non-linear processes,
- to experimentally analyse the robustness of neural networks to noisy data such as those that occur in real world process control applications.

2.1. A benchmark neuro-control problem

Many of the limitations of conventional solutions arise in trying to control poorly modelled non-linear systems, e.g. unrealistically assuming that there is no noise in the measurements and that the control variables (e.g. flow rate) takes on the exact value that the controller requires. Using a pattern recognition problem, a neural networks capability in dealing with such control issues has been practically investigated.

In the benchmark discussed in this section, a large number of patterns consisting of fluid flow readings from a relatively simple process control rig developed by Bytronic Ltd (UK) have been recorded. Each input training pattern consists of the fluid flow readings in litres taken at five second intervals over a period of one minute. The patterns can be divided into three broad categories representing the state of the system output:

- stable fluid flow,
- unstable fluid flow,
- transition from one flow level to another.

Various networks were constructed and used to recognise the three categories of patterns using an ANN system supplied by Science Applications International Corporation (USA) which was run on a 486 microprocessor. The outputs of such networks were used as inputs to a multi-input neural controller, enabling the controller to take account of the exact state of the flow rate. The learning and generalising ability of these networks, particularly in a noisy environment (in which conventional systems perform poorly), were systematically examined and compared.

Many real world processes are dynamic non-linear systems. Recurrent neural networks appear to be ideally suited for modelling non-linear systems. As part of the experiments discussed in this paper, a number of recurrent BP networks were designed and applied to the problem. Their contributions in process control applications were then evaluated.

2.2. Appropriate network

There are at present no fixed rules available for choosing the right network for a given application. One way of finding the most appropriate network is to decide upon the training procedure required for the application. Since in the application discussed here, we define the exact output for every input in the training set, supervised training is the most suitable. Hence, networks such as the multi-layer feed-forward network with back-propagation (also referred to as the back-propagation network) and the Boltzmann machine (and its refinements) were likely candidates capable of mapping and producing new outputs. Counter propagation was also a likely candidate because in training, the network receives the input and the target vector.

As part of the original study [7], a large number of experiments, working on the various aspects of applying the most suitable paradigms were carried out. As a result, the network considered most suitable was a multi-layered fully connected feed-forward network with back-propagation. This publication concentrates on the experimental results derived from such networks.

Various aspects of network design (e.g. deciding upon the number of processing elements or the number of layers) were determined experimentally. In other words, a large number of networks were built, trained and tested to determine the most successful configuration. This is the most common method used in practice. Details of the experiments are covered in Section 2.6.

2.3. Data

There are many aspects to consider when developing an optimum network. They include the source, training and testing.

The source

Training data sets for the benchmark were collected from the Process Control rig illustrated in fig. 1. This unit consisted of: (i) A sump; (ii) A pump; (iii) An analogue-to-digital controlled diverter valve; (iv) A cooler; (v) A process tank; (vi) An analogue-to-digital controlled drain valve.

All features on the unit can be controlled in an analogue or a digital mode. The fluid can be pumped at a flow rate of 0-2 litres per minute from the pump flowing either directly to the process tank, or first to the cooler and then to the process tank. The fluid cycle is completed by either an overflow or drainage, using either of the two valves provided. Based on a fluid flow process to control flow and temperature, the rig presents a relatively small but typical process control operation, analogous to those used in the petrochemical industries.

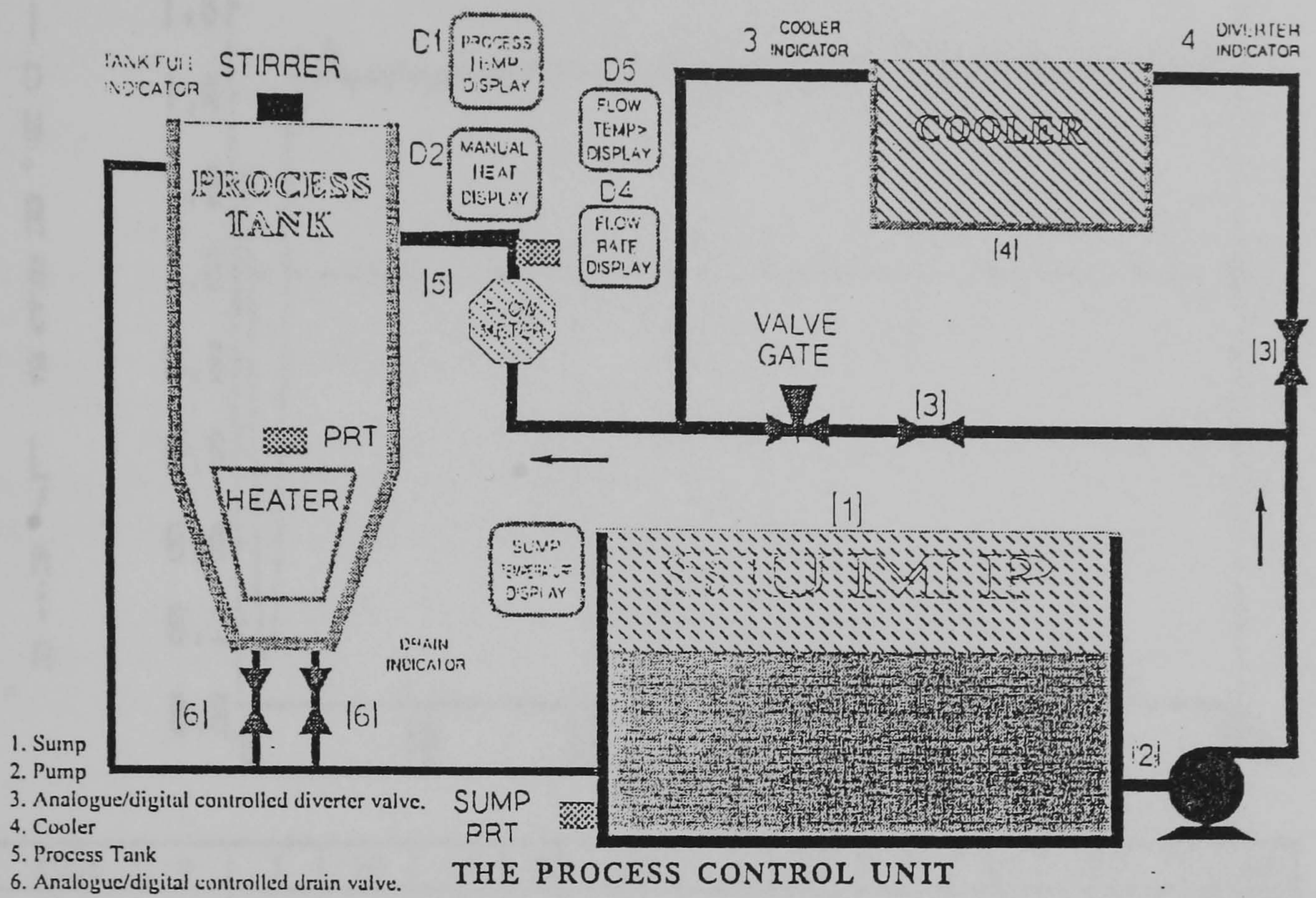
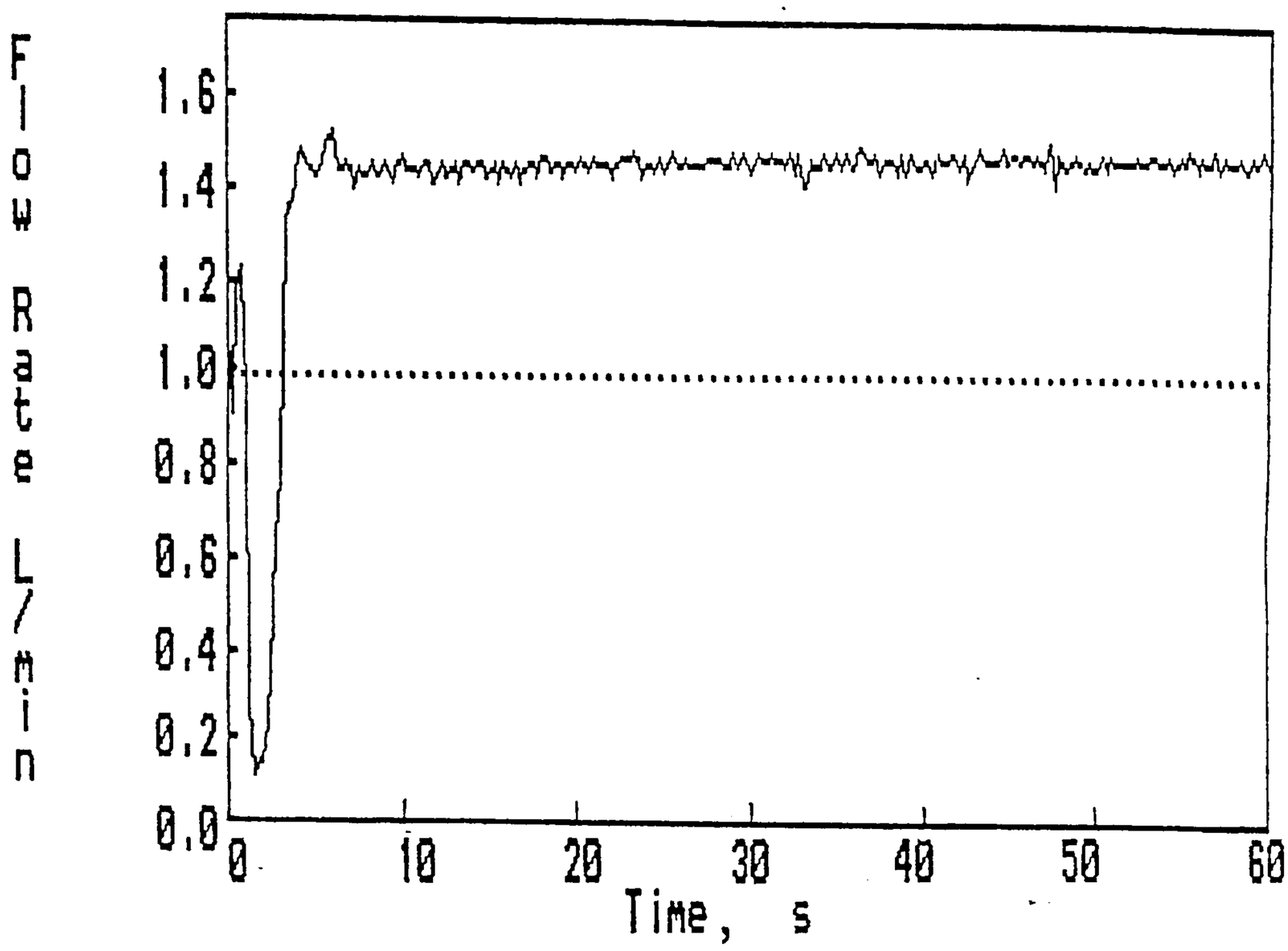


Fig. 1. Schematic diagram of the process control unit used to obtain data on flow rates for training the ANNs

Training

A large number of training sets were developed using a maximum of 120 patterns. As discussed in Section 2.4, with a supervised model such as back propagation, more training examples generally result in a better network response. Each training input pattern represents the fluid flow readings (in litres per minute) taken at 5 second intervals over a period of 1 minute. An example of a flow rate pattern (for manual flow control) is given in fig. 2 together with a digitised data set taken from the flow signal at 5 second intervals. This sampling interval was considered to be an acceptable rate for the process monitoring problem considered. The input patterns can be divided into three categories representing the output flow states of the system, i.e. a steady fluid flow (e.g. 1.5 litres per minute), an unstable flow, or a transition from one flow rate to another.



Time	0	5	10	15	20	25	30	35	40	45	50	55	60
Flow	1.10	1.44	1.18	1.45	1.46	1.45	1.45	1.49	1.47	1.49	1.46	1.48	1.48

Time [s], Flow [l/min].

Fig. 2. Example of a flow rate signal (above) and the data (below) used for training

In general, the change in the fluid flow was set manually, but in some cases set automatically. The sets were then pre-processed before being fed to the system. Pre-processing mostly involves scaling down and normalising the data. Scaling down is used to avoid fluctuations in one input with a much larger range swamping any importance in a second input with a relatively small range. Normalisation is required when the numeric input data has a natural range that is sometimes other than the processing elements operating range.

White noise was added to some of the data. Noise in the form of fluctuations that do not carry information is present in real world applications. Adding white noise to data sets from the process control rig accustoms the networks to the noise typically found in an industrial environment and in addition, provides the network with enhanced data sets. In the experiments conducted for this work, noise was calculated as a pseudo random number from a flat distribution with a maximum amplitude equal to the number entered. This means that, for a noise value of 0.1 for example, each element of each input vector has a random value between ±0.1 added to it independently.

Each training output pattern consists of three numbers representing a steady, unstable or changing state. Such outputs can be used as input signals for a multi-input process controller, enabling it to take account of the state of the flow rate. The values for the training output patterns were chosen on the basis of them being simple to map to their corresponding input fluid flow patterns.

Testing

Networks were tested to evaluate the training process. The most important issues for investigation were as follows:

- Presenting the network with training cases which were slightly different from each other. A well trained network should be able to use its knowledge to recognise new but similar patterns.
- Adding white noise to training data sets to accustom the network to the noise typically found in an industrial environment, and to provide the network with enhanced data sets. Once trained, such a network should be able to correctly identify patterns of this type.
- Testing different network configurations in terms of variables and processing elements. Because of the lack of rigorous theory, a large number of networks were built, trained and evaluated to determine the most successful configuration.
- Evaluating similar networks trained on a variety of training files. The performance of a network is determined by its architecture as well as the quality and quantity of the data used. Thus, to optimise the network, proper recordings and preparation of input data is of utmost importance. In general, it was found that more training data resulted in a more accurate performance by the network when using a supervised model. With an unsupervised model, too many examples can interfere with each other.

Another important consideration is the role of feature detectors. Identification of strong positive or negative weight values operating on a middle layer processing element can help the designer to differentiate between features and detectors.

2.4. Implementation

Architecture

Simple two layered networks can map all representations present in the input data because of the direct mapping of inputs to outputs. However, they fail to learn if the intrinsic representation of the data is non-linearly separable, e.g. the exclusive OR function. By adding intermediate layers, learning such arbitrary mappings is possible. The intermediate layer allows the network to develop its own internal representation of the input patterns. The internal representation and the learning of features not explicitly obvious in the input data is constituted by the activations of the intermediate layers processing elements. They provide the network with an internal representation capability allowing it to decide upon whatever is important in representing the mapping. This is in

contrast to relying on intrinsic relationships already built into the training data in two layer networks.

The first set of experiments given in tab. 1 were concerned with seeking to apply back-propagation techniques to the experimental data. This involved many important changes within the network to find the most suitable configuration. These included varying the number of layers, the number of processing elements in the intermediate layer(s), and the learning parameters. Several networks were developed and then trained on data obtained from the process control rig. The generalised delta rule was used for the networks to learn to develop the required features to perform the desired mappings. The objective of the experiments was to find the most successful configuration to reach a minimum value for the average Root Mean Square (RMS) error in the shortest number of training cycles or iterations.

Sample experimental results demonstrating the various aspects of the process are included in tab. 1. With the exception of network number 5, which has two middle layers containing 6 and 4 processing elements, they all have only one middle layer containing six processing elements. The decision for this configuration was achieved experimentally by varying the number of layers and the number of processing elements in the intermediate layer(s), while keeping all the other elements constant. All the networks were trained on a training file containing 63 patterns collected from the process control rig. To help the networks in their pattern recognition task, the 63 input patterns consisted of similar versions of 12 input training patterns.

At the start of training, weights for all of the 8 networks were initialised to random values between +1.0 and -1.0. Although with such a small range, learning takes longer, this range was considered necessary because the goal was to keep the activations close to 0 at the start of training. With larger values it was found that learning was generally shorter but the average RMS error as well as the maximum output unit error tended to be higher.

Training was halted for networks 1, 2 and 3 after 120000 iterations, as they had clearly established a local minima. Training for the rest of the networks was terminated when their total RMS error reached less than 0.05.

Varying the parameters:

• **Noise.** Noise in the form of fluctuations that do not carry deterministic information is present in nearly all real world applications. In this work, noise was added to the data sets from the process control rig in an attempt accustomise the networks to the noise typically found in a process control environment. Most networks in the experiment were trained using a significant amount of noise (up to 20%) which makes it difficult for a human to recognise the patterns. In experiments conducted to date, a consecutive decay rate (usually around 0.0001) has been set for the noise to reduce gradually (i.e. the noise is reduced by a factor of 0.0001 from each element of each input vector). For example, for a noise value of 0.1, each element of each input vector has a random value between ±0.1 added to it independently; the noise is then reduced gradually by a decay factor (e.g. 0.0001).

• **Learning rate.** True gradient descent in the total RMS error requires that changes are made to the weights only in infinitesimal steps and only after the entire trainingset has been processed. (In order to use less RAM, some implementations of the BP

Table 1. Application of the back-propagation algorithm to find the most suitable network configuration

BP net spec.	1	2	3	4	5	6	7	8
Training criteria	Train. halted after 120000			Train. halted when max.output unit error < 1				
Number of hidden layers	1	1	1	1	2 with similar spec.	1	1	1
Learning rate (input to next layer)	0.9	0.9	0.7	0.1	0.5	0.5	0.3	0.1
Learning rate (hidden to output layer)	0.9	0.9	0.7	0.1	0.5	0.5	0.7	0.1
Learning rate (input to next layer)	0.9	0.5	0.7	0.1	0.5	0.5	0.2	0.9
Momentum term (hidden to next layer)	0.9	0.5	0.7	0.1	0.5	0.5	0.2	0.9
Momentum term (hidden to next layer)	0.9	0.9	0.7	0.1	0.5	0.5	0.7	0.9
Noise (±)	0.1	0.1	0.1	0.1	0.1	0.1	0.2	0.1
Decay	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.001	0.0001
Total RMS error	0.37	0.25	0.28	0.04	0.03	0.04	0.02	0.02
Maximum output error	0.99	0.99	0.99	0.16	0.09	0.09	0.09	0.09
Iterations	12000	12000	12000	1643	374	257	397	176
Testing*	No testing	No testing	No testing	12/18	14/18	12/18	13/18	14/18

* No. of patterns correctly recognised on a testing file containing 18 patterns.

algorithm make weight changes after each pattern.) The proportion of the dictated weight changes may be set as a learning rate. The job of the learning rate constant is to set the speed of convergence of t he weight vector to the ideal value or a more complex error

function in multidimensional space. The larger the learning rate, the larger the weight changes and the faster the learning. The idea is to set the learning rate as high as possible, without causing the RMS error to oscillate significantly. Generally, the optimum value of the learning rate depends on the shape of the error function in weight space. The learning rate constant must be less than 2.0, or the network cannot be stabilised [2]. The learning constant must also be positive. If negative, the direction of the delta vector will be away from the ideal weight vector. In practice, it has been found in this study that it is best to reduce the value of learning rate until the total RMS error shows a general decrease in value with time. Once it is close to the ideal position, even small differences between similar training samples when combined with a large learning constant, can cause the weight vector to move excessively.

• **The momentum term**. One way to increase the learning rate without making the error oscillate is to add a momentum term to the weight change equation. Training proceeds much faster with this term. The term is a variable that determines the proportion of the last weight change that is added into the new weight change. It has the effect of preventing thrashing due to small oscillations in the error.

Experimentally (as can be seen from network number 4), it has been found that a small value for the momentum term (e.g. 0.1) together with a small learning rate causes the network to converge very slowly. In this study, it was eventually concluded that a momentum value significantly higher (e.g. 0.9) together with a learning rate value (e.g. 0.1) provides the best effect to the weight changes (see network number 8).

2.5. Further experiments to consider the more extreme cases

All the networks in tab. 1 were trained on a training file containing 63 patterns which were normalised ("normed") and scaled between ±0.5. This was considered necessary as networks trained on raw data tend to end up in a local minimum. The networks in tab. 1 were also assisted in their pattern recognition task by using input training patterns which consisted of similar versions of only a limited number of input training patterns. As a result, one of the networks (tab. 1, number 8) managed to learn in only 176 iterations.

To consider the more extreme cases, further experiments were undertaken. A principal condition for a network to perform accurately is the quality and proper representation of the training cases. For training the networks considered in tab. 2, a number of data sets covering unusual and boundary cases were recorded. They consisted of 55 patterns for networks 9 and 10, 110 patterns for networks 11, 12 and 13, and 120 patterns, representing the breadth of the problem, for networks 14, 15 and 16. Network performance and test results presented in tab. 2 are discussed in the following section.

2.6. Performance and analysis of BP models on test data

Training patterns

Neural networks learn relatively better when their inputs are simplified. This is demonstrated by the networks in tab. 1. Training files for all these networks were

Table 2. Application of the back-propagation algorithm to data covering more extreme cases

BP net spec.	9	10	11	12	13	14	15	16
Number of hidden layers	2 with similar spec.	2 with similar spec.	2 with similar spec.	1 hidden layer	2 with similar spec.	1 hidden layer	2 with similar spec.	2 with similar spec.
Training data	Raw train.*	55 patt.**	110 patterns normed & scaled (±)0.5			120 patterns normed & scaled (±)0.5		
Learning rate (input to next layer)	0.5	0.5	0.5	0.1	0.1	0.1	0.5	0.1
Learning rate (hidden to output layer)	0.5	0.5	0.5	0.1	0.1	0.1	0.5	0.1
Momentum term (input to next layer)	0.5	0.5	0.5	0.9	0.9	0.9	0.5	0.9
Momentum term (hidden to output layer)	0.5	0.5	0.5	0.9	0.9	0.9	0.5	0.9
Total RMS error	0.4	0.04	0.04	0.04	0.05	0.02	0.03	0.03
Maximum output error	0.69	0.09	0.09	0.09	0.1	0.09	0.09	0.09
Iterations	120000	708	545	1139	358	1192	960	618
Testing***	Halted ****	12/18	15/18	14/18	15/18	15/18	18/18	18/18

* Raw training data with 55 patterns. ** 55 patterns scaled (±)0.5. *** No. of patterns correctly recognised on a testing file containing 18 patterns. **** Halted after 120000 iter.

normalised and scaled between (±)0.5. Networks such as number 9 in tab. 2 which were trained on non-normalised data mostly failed to train. Others (e.g. network number 10) trained on data which were only scaled within a certain range, still had difficulty in learning. Networks 1 to 8 in tab. 1 were additionally assisted in their "pattern recognition" learning task by using input training patterns which consisted of similar versions of only a dozen input training patterns. As a result, one of the networks (number 8) managed to learn in only 176 iterations. However, when tested on unseen data they all performed poorly. For

example, in the case of network number 8, it failed to recognise 4 out of 18 test patterns. This is because an adequate number of training patterns was not available to allow the networks to cover the problem domain.

To find the least number of inputs to adequately represent the problem, the networks in tab. 2 were trained on sequentially larger numbers of different input patterns. As can be seen from the results in tab. 2, training took longer, but performance on testing was improved. Networks 14, 15 and 16 which were trained on 120 different patterns performed relatively well. Networks 15 and 16 managed to correctly recognise all of the test patterns, from a test file containing 18 unseen patterns.

Architecture

Choosing the right number of hidden processing elements and layers for the application has been carried out through a process of trial-and-error. The process was time consuming, tedious and at times contradictory. For example, increasing the number of hidden processing elements generally resulted in longer learning. Sometimes, however, increasing the numbers slightly actually resulted in decreasing learning time. In general, too many processing elements resulted in poor generalisation, i.e. the networks tended to memorise the training set rather than learn it. Defining too few hidden processing elements mostly resulted in incorrect classification. Experimentation eventually lead to an architecture consisting of 6 hidden processing elements for networks with one hidden layer presented in tabs 1 and 2. All the networks with two hidden layers (e.g. networks 15 and 16) have 6 processing elements in the first hidden layer and 4 in the second.

Only a few networks with more than two hidden layers were constructed. It has been shown by many researchers (e.g. Blum and Li, 1991 [1]) that a continuous function can be well approximated by a static neural network with a single hidden layer, where each processing element in the middle layer has a continuous sigmoidal non-linearity. In other words, there is no need to build networks with very many middle layers. Despite such results, further work needs to be done to investigate the relationship between accuracy of the function being approximated with the number of hidden layers. Chester [3] has pointed out that neural networks with two hidden layers appear to provide higher accuracy and better generalisation than a network with a single hidden layer. For the benchmark considered here, this has actually been confirmed, i.e. networks with two hidden layers (e.g. networks 15 and 16 in tab. 2) performed much better than networks with a single hidden layer. To accustom such networks to the noise typically present in a process control environment, a random noise value between ±0.1 was added to each element of each of their input vectors independently. Network 16, managed to train in only 618 iterations. For testing, a file containing 18 unseen new patterns was used. Both networks 15 and 16 managed to correctly recognise all the test patterns.

2.7. Implementing and testing recurrent back-propagation

The networks developed in the previous section are static, feed-forward, or non-recurrent networks. They are able to approximate non-linear functions to a desired degree

of accuracy. As a result, they have been used to model dynamic systems. However, the use of dynamic, feed-back, or recurrent networks to represent dynamic systems may be more appropriate. A recurrent network may be able to do a lot more than a back-propagation network with the same number of weights [6]. No theorem confirming this yet exists. The feed-back in recurrent networks implies local memory characteristics and may cause instability. In a stable network, successive iterations produce smaller and smaller output changes until eventually the outputs become constant. In an unstable network, the process may never end. Chaotic systems, which have interesting properties of their own, are one example of unstable networks.

In the recurrent back-propagation architecture used for this application, in addition to the feed-forward connections from the input layer to the hidden layer and from hidden layer to the output layer, feed-back connections were made between hidden units and between output units. All the processing elements in these layers have feed-back connections to themselves and to every other processing element in that layer. As a result, the generated outputs depend not only on the current inputs but also on the current state of the units in layers with feed-back connections. The feed-back loops in the output layer are only there to allow the network to learn relationships between successive output vectors in a sequence (as potentially such a relationship can exist in the benchmark). More importantly, however, are the feed-back loops in the hidden layer which allow the network to form a "memory trace" of the input sequence. Such architectures are ideally suited for control and modelling of the forward or inverse dynamics of a chemical process control system. This is because such systems are themselves non-linear dynamical systems.

The experiments compounded in tab. 3 are concerned with seeking to apply the recurrent back-propagation network to the experimental data within a noisy environment. Networks 17 and 18 were trained on data consisting of 110 patterns representing the problem. As can be seen from the results, network number 17 did not do well on testing. Network 18, with a more suitable configuration for the application, performed slightly better. Networks 19, 20 and 22 which were trained on a data file containing 120 patterns performed much better.

Using a testing file containing 18 unseen patterns, network 19 managed to correctly recognise 16 patterns. Networks 20 and 22 have the same configuration as 19. They were trained on the same training file as 19, but in a much noisier environment (starting with 10% noise rather than 1% noise). As a result, they managed to correctly recognise all the test patterns. Network 20 managed the training in 1988 iterations. To improve upon this noise was reduced in larger steps for network 22, i.e. a decay rate of 0.02 was used rather than 0.01. As a result, the number of iterations for network 22 was reduced to 832. The stopping criteria for training the networks 20 and 22 was for the value of "the maximum output unit error" to reach 0.1 or less. By increasing this value to 0.2, these networks managed to recognise the patterns satisfactorily after training for a significantly reduced number of iterations. These results are presented in columns tab. 3. From the results given in tab. 3, it is clear that network 23 correctly recognised all the unseen test patterns after training for only 205 cycles.

Table 3. Application of recurrent back-propagation to data

BP recurrent networks	17	18	19	20	21*	22	23**
Noise (±)	0.01	0.01	0.01	0.1	0.1	0.1	0.1
Decay	0.001	0.001	0.001	0.001	0.001	0.002	0.002
Training data	110 patterns normed & scaled (±0.5)		120 patterns normed & scaled (±0.5)				
Learning rate constant (input to next layer)	0.2	0.1	0.1	0.1	0.1	0.1	0.1
Learning rate constant (hidden to output layer)	0.2	0.1	0.1	0.1	0.1	0.1	0.1
Momentum term constant (input to next layer)	0.8	0.9	0.9	0.9	0.9	0.9	0.9
Momentum term constant (hidden to output layer)	0.8	0.9	0.9	0.9	0.9	0.9	0.9
Total RMS error	0.05	0.04	0.04	0.03	0.04	0.03	0.04
Max. output unit error	0.1	0.09	0.1	0.09	0.19	0.09	0.19
Iterations	1231	1381	1962	1988	461	832	205
Testing***	13/18	14/18	16/18	18/18	18/18	18/18	18/18

* The same as net 20 but train. halted earlier.

** The same as net 22 but train. halted earlier.

*** No. of patterns correctly recognised on a testing file containing 18 patterns.

By applying noise randomly to each input pattern just before it is presented to the network, the network operates on a series of approximations to the input. Such a technique forces the network to generalise; this is one of the key goals in the applications of neural networks. As tab. 3 demonstrates, using slightly noisy data effectively improves the training time and learning. This result demonstrates the importance of noise in training, which is in sharp contrast to conventional systems, and demonstrates the neural networks potential in dealing with noisy data that occurs in process control engineering.

Discussion

This section has discussed experimental results and some of the main considerations in the process of deciding upon a suitable configuration for the benchmark. Applying a small amount of noise randomly to each input pattern actually forces the networks to generalise more readily. In process control applications such robustness to noisy data is a principal issue. Networks with two intermediate layers perform particularly well. To represent dynamic systems, feed-forward networks can be extended to include feed-back connections. Such dynamic models have been developed by adding feed-back connections between hidden units and between output units of the feed-forward networks. Dynamic networks are able to do a lot more than a back-propagation network with the same number of weights [6]. Discussions on the networks which performed well, and some general conclusions for all the experiments undertaken in this study are presented in the following section.

3. Analysis of experimental results

Section 2 described a benchmark in which, using data collected from a process control rig, neural networks modelled the underlying relationships of the training set in a noisy environment. Such characteristics are of enormous importance in process control and possibly show their greatest area of promise. This section further discusses the benchmark network performance and the experimental results presented in Section 2.

3.1. The benchmark

In the benchmark described in Section 2, patterns consisting of fluid flow readings from a process control rig were gathered. The patterns could be divided into three broad categories representing the state of the system output. Various neural paradigms were applied to the patterns to recognise the categories. The outputs of such networks could be used as inputs to a multi input neural controller, enabling the controller to take account of the exact state of the flow rate. The general steps taken in the development of the networks for the process control application were as follows:

- creating various network structures,
- gathering and preparing training data,
- performing training,
- comparing performances,
- gathering and preparing new data for testing,
- performing testing,
- analysing and applying the results.

On completion of these stages, the networks could be embedded in stand-alone process control applications.

3.2. On using the back-propagation learning rule

During training, a learning rule is required for systematically updating the weights of a network in response to input signals and the values supplied by the transfer function. The objective of learning depends on the application. In the benchmark considered here, the primary objective was pattern recognition and the learning rule was required to classify sample data and predict successfully using new data in a noisy environment. When controlling a largely unknown system, the objective of learning is to approximate non-linear functions and/or to make the system follow the desired response.

Learning rules are broadly categorised into to two types:

• "Supervised" learning rules such as error based approaches including stochastic Boltzmann machines and error corrections, i.e. back-propagation and least mean square paradigms.

• "Unsupervised" or output-based paradigms such as the Hebbian and competitive learning rules. In the supervised learning paradigms, training data files contain sets or sequences of vector pairs used for training a network. Each vector pair is composed of an input vector and a target vector. In the application considered for this paper and since we have defined the exact output for every input in the training set, supervised networks were suitable candidates and a number of them were applied to the experimental data as discussed in Section 2.

The network of choice was a multi-layered fully connected feed-forward network trained with back-propagation. Sample results applying back-propagation techniques to data sets covering unusual and boundary cases are highlighted in tab. 4. To accustom the networks to the noise typically found in a process control environment, random noise in the range ±0.1 was added to the training data sets.

Table 4. Application of the back-propagation algorithm to experimental data.
This table is a reduced version of tab. 2.

Networks	12	13	14	16
Hidden layers	1	2 (similar)	1	2 (similar)
Training patterns	110 normed & scaled		120 normed & scaled	
Iterations	1139	358	1192	618
Testing (patterns recognised)	14/18	15/18	15/18	18/18

Training for all the networks was terminated when the total RMS error reached less than 0.05 and the maximum output error reached less than 0.1. At the start of training, weights for all the networks were initialised to a range between -1.0 and +1.0. With larger values it was found that learning was generally shorter but the average RMS error as well as the maximum output unit error tended to be higher. To find the least number of inputs to adequately represent the problem, networks were trained on sequentially larger numbers of different input patterns; the final training data consisting of 120 different patterns.

As shown in tab. 4, networks with two hidden layers generally appeared to provide higher accuracy and better generalisation than those with a single layer. Network number 16, which has two hidden layers with similar specifications, managed to correctly recognise all the test patterns after only 618 iterations. When tested on unseen data sets with noise levels of 0.1, 0.15 and 0.2 added to them, the network managed to recognise 18/18, 17/18 and 16/18 respectively. All the networks with 2 hidden layers have 6 processing elements in the first hidden layer and 4 in the second. More hidden processing elements generally resulted in poor generalisation, i.e. the networks tended to memorise the training set rather than learn it. Defining too few hidden processing elements mostly resulted in incorrect classification.

3.3. Extending the BP feed-forward networks with feed-back

The multi-layered feed-forward networks in tab. 4 used the Generalised Delta Rule to perform static mappings of input data to output data. By allowing extensions in the form of feed-backs of hidden and output units to themselves and to other units in the same layer as illustrated in fig. 3, networks may exhibited sequential behaviour. In other words, their generated output depends both on the current inputs and on the current state of the units in layers with feed-back connections. However, there is no well established learning rule currently known for such a recurrent network. In fact the theory of dynamic neural networks (compared with static networks) is in general, not well developed. Hence, Rumelhart's method [9] was first used to produce an equivalent static network. The Generalised Delta Rule was then applied to train the network. The main drawback with unwinding a recurrent network in time to create an equivalent feed-forward network is the memory resources required.

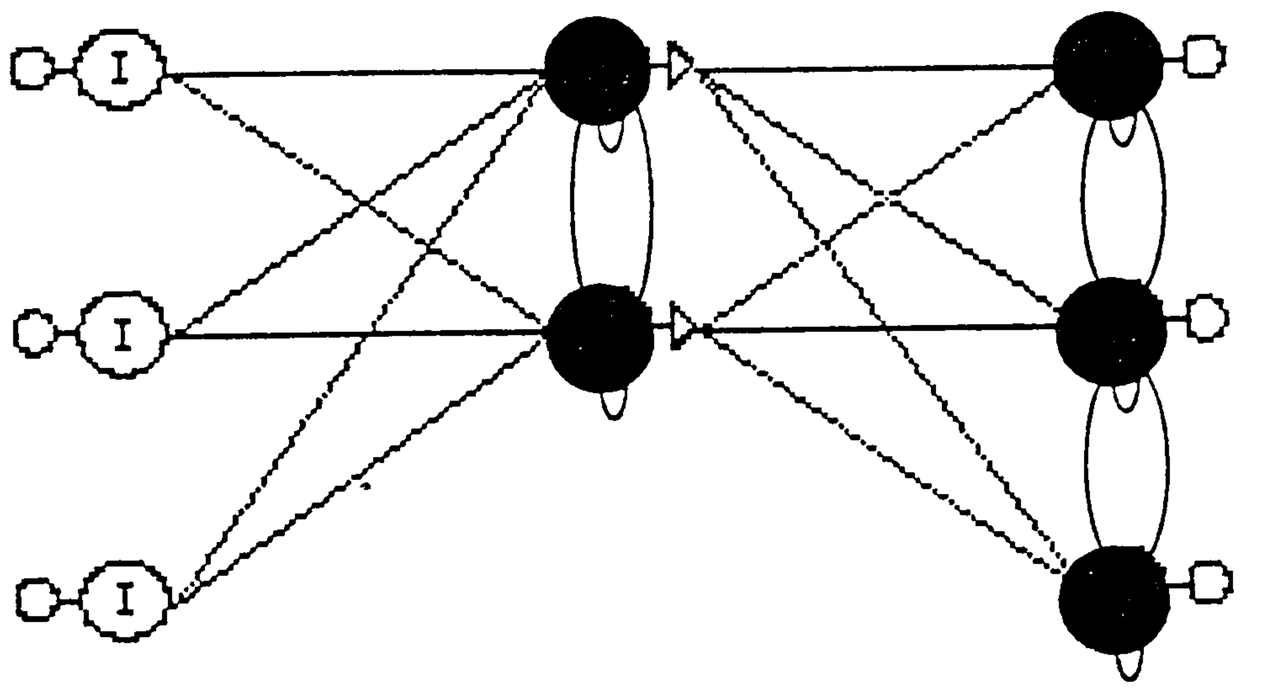


Fig. 3. A three layer recurrent network

Sample experiments seeking to apply the recurrent back-propagation network to the experimental data within a noisy environment are included in tab. 5. This table highlights some of the dynamic networks presented in tab. 3 which in addition, includes the various learning parameter values used.

Table 5. Application of recurrent back-propagation to data
This table is a reduced version of tab. 3

Networks	19	20	22	23	24	25
Training data	120	120	120	120	120	120
Noise (±)	0.01	0.1	0.1	0.1	0.2	0.2
Decay	0.001	0.001	0.002	0.002	0.05	0.05
Iterations	1962	1988	832	205	stopped 1000 local min.	340
Testing with 0.0 noise	16/18	18/18	18/18	18/18	not applicable	16/18
Testing with ±0.1 noise	13/18	18/18	18/18	18/18	not applicable	15/18

The stopping criterion for training networks 19, 20 and 22 was based on the networks reaching a maximum output error value of 0.1 or less. This value for networks 23, 24 and 25 was increased to an acceptable value of 0.2. All the networks have the same configurations, but were trained using different noise levels. At the start of training, weights for the networks were initialised to values between -0.1 and +0.1.

Using a testing file containing 18 unseen patterns with no noise added, network 19 managed to correctly recognise 16 patterns. Network 20, which was trained in a much noisier environment than 19 (the level of noise starting at 0.1 rather than 0.01) correctly recognised all the test patterns. Thus, network 20 actually used the additional noise to improve its performance.

To reduce the number of iterations of network 20, the added noise was reduced in larger steps in network 22, which is otherwise similar to 20, i.e. a decay rate of 0.02 rather than 0.01 was used for network 22. As a result, the number of iterations for 22 was reduced to less than half of that of 20 (from 1988 to 832).

By increasing the stopping criteria (i.e. the maximum output unit error value) from 0.1 to 0.2, network 23, which is otherwise the same as 22, managed to correctly recognise all the test patterns after only 205 iterations. Network 25 started training with a significant amount of noise (0.2). Noise was then reduced in much larger steps than in the case of network 23 (i.e. 0.05 compared with 0.002). The network managed to recognise 16 patterns after 340 iterations. It is clear, that in general, a large amount of noise (e.g. more than 0.15) and/or a reduction of noise in large steps causes instability, i.e. training in such an environment does not necessarily produced smaller and smaller output changes. For example, training for network 24 which is exactly the same as 25 was halted after 1000 cycles, as it had clearly reached a local minimum.

When tested on unseen patterns with a noise value between -0.1 and +0.1 added at random, correctly trained networks (e.g. network 23) could recognise all the patterns. However, networks generally showed a degradation in performance with any increase in the noise level. For example, network 25 which recognised 16 out of 18 patterns with no noise added, only recognised 15 patterns, when a random noise between -0.1 and +0.1 was added to the test patterns. On increasing the amount of noise to ±0.15, the network only managed to recognise 14 patterns.

In general, training these networks proved to be more problematic than that of the static networks presented in tab. 1. Using slightly noisy data effectively improved the learning and the training cycles. With specific reference to their application in control systems, such dynamic networks are particularly appropriate for system modelling identification and control. However, the general theory with regard to learning rules and architecture needs to be further developed.

4. Conclusions and discussion

The study reported in this paper has shown that ANNs may be suitable for solving many process control engineering problems particularly in modelling control applications in a noisy environment. They have already been used successfully as non-linear filters to extract signals from noise. In contrast, conventional approaches suffer from sharp increases in computation and implementation costs with non-linearity. As a result, such approaches (e.g. the general lest-mean-square method extended on the basis of a truncated Wiener series) have only been used in weak non-linear cases. Rather than filtering the noise as a pre-processing operation, this work has investigated and analysed the effect of adding noise. To accomplish this, two principal objectives were set, namely, to address process control issues regarding noise and to investigate and evaluate mapping and generalising attributes of various paradigms for neuro-control solutions.

These objectives have been achieved and in doing so have provided the following important conclusions:

- The whole process of applying feed-forward neural networks to a process control application, particularly in a noisy environment, has been experimentally analysed. Such networks are inherently stable as no information is fed back during operation.
- By extending the static networks with feed-back connections, the utility of dynamic networks in neuro-control applications were practically examined. When conducting the experiments, it was found that training these networks with feed-back is more problematic (e.g. in terms of stability). However, such recurrent networks offer a great potential for further research in process control applications.
- Neural networks robustness to noisy data has been analysed. By adding a significant amount of noise to each element of all the input training vectors independently, a robust weight set was developed. On testing, it was found that careful use of slightly noisy data effectively improved the networks performance in terms of generalisation and learning as well as causing a reduction in the number of training cycles. Adding a small

amount of noise randomly, forced the networks to train on a series of approximations to the input patterns. This drove the networks to generalise, resulting in an improvement in their performance. In other words, the added noise actually helped the networks in their pattern recognition task.

From the neuro-control viewpoint, such robustness to noisy data that occurs in real world applications is an important attribute. It indicates the neural networks potential in solving problems where conventional solutions have proved inadequate, if not impossible. A reasonable conclusion to be drawn from the literature surveyed, as well as the experimental work carried out for this paper is that process control as a promising area for further analysis and development of neuro-control systems.

Acknowledgement

The authors would like to thank Bytronic Ltd (UK) and the Department of Chemistry at De Montfort University, Leicester for helping generate the data. This research has been partly sponsored by Fisher-Rosement International (previously Fisher Controls Ltd).

References

1. **Blum E.K., Li L.K.:** Approximation theory and feed-forward networks. *Neural Networks*, **4** (1991), 511-515.
2. **Caudill M., Butler C.:** Understanding neural networks. The MIT Press, Cambridge, MA 1991.
3. **Chester D.:** Why two hidden layers are better than one. Proc. IEEE Int. Joint Conf. Neural Networks, 1992, 265-268.
4. **Declaris N., Su M.:** A novel class of neural networks with quadratic functions. IEEE/SMC, IEEE Catalogue No. 91CH3067-6, New York 1991.
5. **Gallent, A.R., White H.:** There exists a neural network that does not make avoidable mistakes. Proc. IEEE Conf. Neural Networks, vol. 1, San Diego 1988, 657-664.
6. **Hecht-Nielsen R.:** Neurocomputing. Addison Wesley, New York 1987.
7. **Osanlou A.:** Neural networks in process control - M. Phil. Thesis, School of Computing Sciences, De Montfort University, Leicester 1996.
8. **Poggio T., Girosi F.:** Networks for approximation and learning. *Proc. IEEE*, **78** (1990) 9, 1481-1497.
9. **Rumelhart D.E., Hinton G.E., Williams, R.J.:** Learning internal representations by error propagation. *Parallel Distributed Processing*, vol. 1, MIT Press, 1986.
10. **Watson G.A.:** Approximation theory and numerical methods. J. Wiley & Sons, New York 1980.
11. **Wasserman P.D.:** Advanced methods in neural computing, Van Nostrand Reinhold, New York 1993.

REFERENCES AND BIBLIOGRAPHY

1. **Äström, K.J. and Wittenmark, B. [1989],** *Adaptive Feedback Control*, New York: Addison Wesley.

2. **Äström, K.J. and McAvoy, T.J. [1992]**, Intelligent Control: an Overview and Evaluation. In *Handbook of Intelligent Control*, White, D.A. and Sofge, D.A., Eds, New York: Van Nostrand Reinhold.

3. **Agrawal, P., Lee, C., Lim, H.C. and Ramkrishna, D. [1982]**, Theoretical Investigations of Dynamic Behavior of Isothermal Continuous Stirred Tank Biological Reactors, *Chemical Engineering Science*, 37: 453.

4. **Agrawal, M. and Seborg, D.E. [1987]**, Self Tuning Controllers for Non-linear Systems, *Automatica*, 23(2): 209-214.

5. **Albus, J.S. [1975]**, A new approach to manipulator control: The celebellar model articulation controller (CMAC), *Trans. ASME J. Dyn. Sys. Mea. and Control*, 97, 220 - 227.

6. **Albus, J.S. [1979a]**, A Model of the Brain for Robot Control. Part 2, *Byte*, July, 54 - 95.

7. **Albus, J.S. [1979b]**, Mechanisms of Planning and Problem Solving in the Brain *Mathematical Biosciences*, 45, 247 - 293.

8. **Aleksander, I. and Morton, H. [1989]**, *An Introduction to Neural Computing*, London: North Oxford Academic Publishers.

9. **Alkin, O. [1994]**, *Digital Signal Processing: A Laboratory Approach Using PC-DSP*, Prentice Hall Engineering, Science & Math, ISBN 0-13-328139-6

10. **Anderson, A. [1990]**, Ed., *Neurocomputing 2: Directions for research*, Cambridge, MA: MIT Press.

11. **Antoniou. A, [1993]**, *Digital Filters: Analysis, Design and Applications*, McGraw-Hill.

12. **Apolloni, G., Avanzini, N. and Ronchini, G. [1990],** Diagnosis of Epilepsy Via Back Propagation, *Proc. IJCNN,II*, 571-574.

13. **Atkenson, C.G. and D.J. Reinkensmeyer [1989]**, Using Associative Content Addressable Memories to Control Robots. In *Proc.* of 1989 *IEEE Int. Conf. on Robotics and Automation,* Scottsdale, Arizona, 1859-1864.

14. **Bandt C, Graf S, Zaehle M (Eds.), [1995],** Fractal Geometry and Stochastics, (*Progress in Probability. Vol. 37*), Springer-Verlag, ISBN 3-7643-5263-9.

15. **Barabasi. A, [1995]**, *Fractal Concepts in Surface Grows*, Cambridge: Cambridge University Press.

16. **Barnsley. M, [1993],** *Fractals Everywhere*, London: Academic Press, ISBN 0-12-079061-0.

17. **Barto, A., Sutton, R. and Anderson, C. [1983]**, Neuron-like Adaptive Elements That Can Solve Difficult Learning Control Problems, *IEEE Trans. Systems, Man and Cyber., SMC* - 13 (5).

18. **Barto, A.G. [1990]**, Connectionist learning for control. In *Neural Networks for Control*, Miller, W.T., Sutton, R.S. and Werbos, P.J. [Eds.], London: The MIT press.

19. **Bhat, N.V. and McAvoy, T.J. [1990]**, Use of Neural Nets for Dynamical Modelling and Control of Chemical Process Systems, *Computers Chem Eng.,* 14, 573-5583.

20. **Bhat, N.V., Minderman, P.A., McAvoy, T. and Wang, N. S. [1990]**, Modelling Chemical Process Systems Via Neural Computation, *IEEE Control Systems Magazine,* 10, 24-29.

21. **Biglieri E, Luise M (Eds.), [1996],** Signal Processing in Telecommunications *(Proceedings of the 7th International Thyrrhenian Workshop on Digital Communications*, Viagreggio, Italy, September 10 - 14, 1995), Springer-Verlag, ISBN 3-540-76019-9.

22. **Bishop, J.M. and Torr, P. [1992],** The Stochastic Search. In *Neural Networks for Image, Speech and Natural Language*, Lingard, R. and Nightingale, C. [Eds.], Chapman Hall.

23. **Blanchman N. M. [1996],** *Noise and its Effect on Communication*, New York, London, McGraw-Hill.

24. **Blackledge J. M. [1993],** On the Synthesis and Processing of Fractal Signals and Images, in '*Applications of Fractals and Chaos'*, New York: Springer-Verlag.

25. **Blackledge J M, [1993],** Applications of the Fractal Geometry to Pattern Recognition in Digital Images, *SERC Technical Monograph,* Institute of Simulation Sciences, De Montfort University.

26. **Blackledge J M, Foxon, B. and Mikhaliov, S. [1996]:** Random Fractal Coding Techniques, *SERC Technical Monograph,* Institute of Simulation Sciences, De Montfort University.

27. **Blackledge J M, Turner M J, Andrews P R, [1998],** *Fractal Geometry Digital Imaging*, Academic Press, ISBN 0-12-703970-8.

28. **Blum, E.K. and Li, L.K. [1991]**, Approximation Theory and Feedforward Networks, *Neural networks*, vol. 4, 511-515.

29. **Bulsara, A. R. and Gammaitoni, L. [1996],** *Phys. Rev.*, E48, 3390.

30. **Bunde A, Havlin S (Eds.), [1994]**, *Fractals in Science*, Springer-Verlag, ISBN 0-387-56221-4.

31. **Bunde, A. and Havlin, S. (Eds.), [1996],** *Fractals and disordered systems*, 2nd edition, Springer-Verlag, ISBN 0-387-56219-2.

32. **Burrus, C. S., Gopinath, R. A., and Guo, H. [1997],** *Introduction to Wavelets and Wavelet Transforms: A Primer*, Prentice Hall Engineering, Science & Math, ISBN 0-13-489600-9.

33. **Burrus C. S. and Parks T. W. [1985],** *DFT/FFT and Convolution Algorithms*, Wiley Inter-Science.

34. **Cacoullos, T. [1966],** Estimation of a mulivariate density, *Ann. Inst. Statist. Math.* Vol. 18, pp. 179-189.

35. **Casdagli, M. [1989]**, Non-linear System Prediction of Chaotic Time Series, *Physica D.*, 35, 335-356.

36. **Caudill, M. and Butler, C. [1992]**, *Understanding Neural Networks*, Cambridge, Massachusetts: The MIT Press.

37. **Chen, S., Billings, S. A. and Grant, P.M. [1990]**. Non-linear System Identification Using Neural Networks, *Int. J. Control,* 51, 1191-1214.

38. **Cherbit. G, (Ed.), [1987]**, *Fractals: Non-integral Dimensions and Applications*, Wiley, ISBN 0-471-92798-8.

39. **Chester, D. [1990]**. Why Two Hidden Layers Are Better Than One, in *Proc. IEEE Int. Joint Conf. Neural networks [IJCNN],* 265-268.

40. **Chi, S. R., Shoureshi, R. and Enorio, M. [1990]**, Neural Networks for System Identification. *IEEE Control Systems Magazine,* 10, 31-34.

41. **Ciliz, M. K. and Isik, C. [1989]**, Time Optimal Control of Mobile Robot Motion Using Neural Nets. In *IEEE Int. Symposium on Intelligent Control* 1989, 368-373.

42. **Cízek. V. [1986],** *Discrete Fourier Transforms and their applications,* Adam Hilger Ltd,

Bristol and Boston.

43. **Clarke, K. [1988],** *Scale based simulation of topographic relief*, The American Cartographer, 18(1):27-47.

44. **Clarkson, P. M. [1995],** *Signal Processing Methods for Audio, Images and Telecommunications*, Academic Press, ISBN 0-12-175790-0.

45. **Cotter, N.E. [1990]**, The Stone - Weierstrass Theorem and its Application to Neural Networks, *IEEE Trans. on Neural Networks*, vol. 1, No. 4, 290 - 295.

46. **Crownover, R. M. [1995],** *Introduction to fractals and chaos*, Boston: Jones & Bartlett

47. **Davies, E. R. [1993],** *Electronics, Noise and Signal Recovery*, London: Academic Press.

48. **Declaris, N. and Su, M. [1991]**, A Novel Class of Neural Networks With Quadratic Functions. In *IEEE/SMC, IEEE Catalogue No. 91CH3067-6*, New York: IEEE.

49. **Dhawan, A.P. and Arata, L. [1993]**, Segmentation of Medical Images Through Competitive Learning, *The IEEE Int. Conf. on Neural Networks,* vol III, 1277-1288. Sanfrancisco, CA.

50. **Diamond, M.C. [1985]**, *The Human Brain Coloring Book*, New York: Barnes and Nobel Books.

51. **Dietz, W.E., Kiech, E.L. and Ali, M. [1989]**, Jet and Rocket Engine Fault Diagnosis in Real Time, *J. Neural Network Computing*, 1, 1, 5-18.

52. **Dorato, P. and Yedavalli, R.K. Eds., [1990]**, Recent *Advances in Robust Control*, New York: IEEE Press.

53. **Duhamel, P. and Vetterli, M. [1990],** Fast Fourier Transforms: A tutorial review and state of the art, *Signal Processing*, 19, pp. 259-299.

54. **Ebert, D. S., Musgrave, M. and Peachey, D. [1994]**, *Texturing and Modeling*, Academic Press, ISBN 0-12-228760-6.

55. **Edgar, J.[1990]**, *Measure, Topology and Fractal Geometry*, Springer-Verlag, ISBN 0-387-97272-2.

56. **Edgar, G. A. [1997],** *Integral, Probability, and Fractal Measures*, Springer-Verlag, ISBN 0-387-98205-1.

57. **Ersu, E. and Tolle, H. [1984]**, A New Concept for Learning Control Inspired by Brain Theory, *Proc. 9th World Congress of IFAC,* 7, 245-250.

58. **Elliott, D. [1987],** *Handbook of Digital Signal Processing: Engineering*, Academic Press,

ISBN: 0-12-237075-9.

59.**Embree, P. [1995]**, *C Language Algorithms for Real-Time DSP*, Prentice Hall Professional Technical Reference, ISBN 0-13-337353-3.

60.**Ersoy, O. K. [1996],** *Fourier Related Transforms, Fast Algorithms and Applications*, Prentice Hall Professional Technical Reference, ISBN 0-13-624412-2.

61.**Falconer, K. [1990]**, *Fractal Geometry, mathematical foundations and applications*, Wiley, ISBN 0-471-92287-0.

62.**Falconer, K. J. [1997],** *Techniques in fractal geometry*, Wiley, ISBN 0-471-95724-0.

63.**Feuer, A. and Goodwin, G. C. [1996],** Sampling in Digital Signal Processing and Control, *Systems and Control: Foundations and Applications*. Ed.: C.I. Byrnes, Springer-Verlag, ISBN 0-8176-3934-9.

64.**Figueiras-Vidal, A. R. [1996],** *Digital Signal Processing in Telecommunications*, Springer-Verlag, ISBN 3-540-76037-7.

65.**Fu, K.S. [1970]**, Learning Control Systems Review and Outlook, *Trans. IEEE on Aut. Control,* 16, 210-221.

66.**Gallent, A.R. and White, H. [1988]**, There Exists a Neural Network That Does Not Make Avoidable Mistakes. In *Proc. IEEE Conf. Neural networks*, vol. I, 657 - 664, San Diego.

67.**Garcia, G.E. and Morari, M. [1982]**, Internal Model Control 1, A Unifying Review and Some New Results, *Ind. Eng. Chem. Proc.,* Des. Dev., 221:308.

68.**Geng, Z. and Jamshidi, M. [1988]**, Expert Self-learning Controller for Robot Manipulator, In *Proceedings of the Twenty Seventh IEEE Conf. on Decision and Control*, Austin, TX, IEEE Control System Society.

69.**Girosi, F., Poggio, T. and Caprile, B. [1991]**, Extensions of a Theory of Networks for Approximations and Learning: Outliners and Negative Examples. In *Advances in Neural Information Processing Systems 3*, Lippman, R.P., Moody, J.E. and Touretzky, D.S. [Eds.], 750-756. San Mateo, CA: Morgan Kaufmann.

70.**Glanz, F.H., Miller, W.T. and Kraft, L.G. [1990]**, CMAC: An Associative Neural Network Alternative to Backpropagation, *Proc. of the IEEE*, 78[10], 1561 - 1567.

71.**Goodwin, G.C. and Sin, K.S. [1984]**, *Adaptive Filtering Prediction and Control,* Englewood Cliffs, NJ: Prentice-Hall.

72.**Grossberg, S. [1982]**, *Studies of Mind and Brain*, Reidel.

73.**Grossberg, S. and Kuperstein, M. [1986]**, Neural Dynamics of Adaptive Sensory Model Control: *Ballistic Eye Movements*, Amsterdam: Elsevier.

74.**Guez, A., Elibert, J. L. and Kam, M. [1988]**, Neural Network Architecture for Control, *IEEE Control Systems Magazine,* 8, 22-25.

75.**Gulati, S., Iyengar, S.S., Protopopescu, V., Barhen, J. and Toomarian, N. [1987]**, Non-linear Neural Networks for Deterministic Scheduling. In *IEEE First Int. Conf. on Neural Networks*, vol. IV, 687-696.

76.**Gupta, M.M. [1986]**, Ed., *Adaptive Methods for Control System Design*, New York: IEEE Press.

77.**Hammerstorm, D. [1993 a],** Working with neural networks, *IEEE Spectrum,* vol 30, no 6, pp. 26-32.

78.**Hammerstorm, D. [1993 b],** Working with neural networks, *IEEE Spectrum,* vol 30, no 7, pp. 46-53.

79.**Hartman, E.J., Keeler, J.D. and Kowalski, J.M. [1990]**, Layered Neural Networks with Gaussian Hidden Units as Universal Approximations, *Neural Computation* 2[2]: 210-215.

80.**Haykin, S. [1995],** *Adaptive Filter Theory*, 3/e, Prentice Hall Engineering, Science & Math, ISBN 0-13-322760-X.

81.**Hebb, D.O. [1961]**, *The Organisation of Behaviour*, New York: John Wiley.

82.**Hecht-Nielsen, R. [1987]**, *Neurocomputing*, New York: Addison Wesley Publishing Company.

83.**Hinton, G.E. and Anderson, J.A. [1981]**, Eds., *Parallel Models of Associative Memory*, Hillsdale, NJ: Erlbaum.

84.**Hinton, G.E. and Sejnowski, T.J. [1986]**, Learning and Relearning in Boltzmann Machines. In *Parallel Distributed Processing*, vol. 1, Bradford Books, The MIT Press.

85.**Hitachi Ltd. [1990]**, Control Systems for Steel Rolling, *J. of Japanese New Materials Advanced Alloys and Metals*, Nov. 26.

86.**Hopfield, J.J. [1982]**, Neural Networks and Physical Systems with Emergent Collective Computational Abilities, *Proc. of the National Academy of Sciences*, 79, 2554-2558.

87. **Hopfield, J.J. [1984]**, Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons, *Proc. of the National Academy of Sciences*, 81, 3088-3092.

88. **Hopfield, J.J. [1990]**, Artificial Neural Networks are Coming, *IEEE Expert*, An Interview by W. Myers, April, 3-6.

89. **Hopfield, J.J. and Tank, D.W. [1986],** Computing with Neural Circuits: A Model, *Science*, vol. 233, August, 625-630.

90. **Hutchinson, J. [1981],** Fractals and self-similarity, *Indian Journal of Mathematics,* 30, pp. 713-747

91. **Hunt, K., Sbarbaro-Hofer D. and Neumerkel, D. [1993]**, *Neural Control of a Steel Rolling Mill*, IEEE Control Systems, vol. 13, No. 3.

92. **Jang, J. S. R., Sun, T. S. and Mizutani, E. [1997],** *Neuro-Fuzzy and Soft Computing,* Prentice-Hall, Inc., ISBN 0-13-261066-3.

93. **Jamshidi, M., Abdallah, C. and Dawson, D. [1991]**, Survey of Robust Control for Rigid Machines, *IEEE Contr. Syst. Mag.,* Feb. 24-30.

94. **Jardine, L. F. [1993]**, Fractal-Based Analysis and Synthesis of Multispectral Visual Texture for Camouflage, in *'Applications of Fractals and Chaos'*, New York: Springer-Verlag.

95. **Johnson, H. and Dugeon, D. E. [1993],** *Array Signal Processing: Concepts and Techniques*, Prentice Hall Professional Technical Reference, ISBN 0-13-048513-6.

96. **Jordan, M.I. [1988]**, Sequential Dependencies and Systems with Excess Degrees of Freedom. *COINS Technical Report*, 88-27, Depatment of Computer and Information Science, University of Massachusetts, Amherst.

97. **Jordan, M.I. [1990]**, Learning and the Degrees of Freedom Problem, In *Attention and Performance XIII*, M. Jeanneod, [Ed.],. Hillsdale, NJ: Erlbaum.

98. **Jordan, M. I. and Rumelhart, D. E. [1991]**, Forward Models: Supervised Learning with a Teacher. *Occasional Paper #40*, Center for Cognitive Science, Massachusetts Institute of Technology [49 pages].

99. **Josin, G. M. [1990]**, Development of a Neural Network Auto-pilot Model for High Performance Aircraft. In *IEEE Int. Joint Conf. on Neural Networks, IJCNN'90,* 547-550.

100. **Kamas, A. and Lee, E. A. [1988],** *Digital Signal Processing Experiments Using A*

Personal Computer, Prentice Hall Professional Technical Reference, ISBN 0-13-212853-5

101.**Karl, J. [1989]**, *An Introduction to Digital Signal Processing*, Academic Press, ISBN 0-12-398420-3.

102.**Kaye, B. H. [1994],** *A Random Walk Through Fractal Dimensions*, 2nd Ed., Wiley, ISBN 3-527-29078-8

103.**Kaandrop, J. [1994],** *Fractal Modelling, Grows and Form in Biology*, Springer-Verlag, ISBN 0-387-56685-6.

104.**Kawai, T., Yoshiro, M., Makoto, S, and Maki, H.[1993],** Basis of Universal Existance of 1/*f* Fluctuation: Mathematical proof and numerical demonstrations. In Peter H. Handel and Alma L. Chung, Eds, *Noise in Physical Systems and 1/f Fluctuations,* St. Louis: AIP Conf. Proc. 285: 639-642.

105.**Kawato, M., Furukawa, K. and Suzuki, R. [1987]**, A Hierarchical Neural Network Model for Control and Learning of Voluntary Movement, *Biological Cybernetics,* 57: 169-185.

106.**Kawato, M., Uno, Y., Isobe, M. and Suzuki , R. [1988]**, Hierarchical Neural Network Model for Voluntary Movement with Application to Robotics, *IEEE Control Systems Magazine* 8(2): 8-16.

107.**Kawato, M., [1990]**, Computational Schemes and Neural Network Models for Formation and Control of Multi-Joint Arm Trajectory. In *Neural Networks for Control*, Miller, W.T., Sutton, R.S. and Werbos, P.J. [Eds.], Cambridge, MA: MIT Press.

108.**Kohonen, T. [1977]**, *Associative memory: A System Theoretic Approach*, Berlin: Springer-Verlag.

109.**Kohonen, T. [1984]**, Self Organising and Associative Memory, *Springer-Verlag*, Series in Information Sciences, vol. 8, New York: Verlin-Heidelverg.

110.**Kohonen, T. [1988]**, The Neural Phonetic Typewriter, *Computer [IEEE],* vol. 21, No. 3, March, 11- 24.

111.**Kolmogorov, A.N. [1957]**, On the Representation of Continuous Functions of Many Variables by Superposition of Continuous Functions of One Variable and Addition, *Dokl. Akad. Nauk SSSR* 114: 953 - 956, [English Translation: *American Mathematical Society Translations*, vol. 28].

112.**Kraft, L. G. and Campagna, D. P. [1990]**, A Comparison Between CMAC Neural Network Control and Two Traditional Adaptive Control Systems, *IEEE Control Systems*

Magazine, 10, 36-43.

113.**Kumar, S. and Guez, A. [1990]**, Adaptive Pole Placement for Neurocontrol. In *IEEE Int. Joint Conf. on Neural Networks, IJCNN'90,* 397-400.

114.**Kuperstein, M. [1987]**, Adaptive Visual-Motor Coordination in Multijoint Robots using Parallel Architecture. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 1595-1602.

115.**Kurkova, V. [1992]**, Kolmogorove's Theorem and Multilayer Neural Networks, *Neural Networks*, vol. 5, 501 - 506.

116.**Lapedes, A. and Farber, R. [1987]**, Non-linear Signal Processing using Neural Networks: Prediction and System Modelling, *Report LA-UR-87-2662*, Los Alamos National Laboratory.

117.**Lawrence, A.J. and Harris, C.J. [1993]**, A Label-Driven CMAC Intelligent Control Strategy. In *Application of Neural networks to Modelling and Control*, Page, G.F., Gomm, J.B. and Williams, D. [Eds.], London: Chapman and Hall.

118.**Lee, S. and Kill, R.M. [1991]**, A Gaussian Potential Function Network with Hierarchically Self Organising Learning, *Neural networks* 4[2]: 207-224.

119.**Levy Vehel, J., Lutton, E. and Tricot, C. Eds., [1997],** *Fractals in Engineering*, Springer-Verlag, ISBN 3-540-76182-9.

120.**Lightbody, G., Irwin, G.W., Taylor, A., Kelly, K. and McCormick, J. [1994]**, Neural Network Modelling of a Polymerisation Reactor, *Proc. IEEE Int. Conf. Control 94*, vol. 1.

121.**Lim, J. S. [1989],** Two-Dimensional Signal and Image Processing, Prentice Hall Professional Technical Reference, ISBN 0-13-935322-4.

122.**Lippmann, R.P. [1989]**, Review of Neural Networks for Speech Recognition, *Neural Computation*, 1, 1-38.

123.**Ljung, L. [1987]**, *System Identification - Theory for the User.*, Englewood Cliffs, NJ: Prentice Hall.

124.**Ljung, L. and Soderstrom, T. [1983]**, *Theory and Practice of Recursive Identification,* London: MIT Press.

125.**Lotfi-Zadeh, A. [1992],** Fuzzy logic, Neural networks and Soft computing, *Course CS 294-4,* The University of California at Berkeley,

126.**Mandelbrot, B. B. [1977],** *Fractals, Form, Chance and Dimension*, WH Freeman.

127.**Mandelbrot, B. B. [1983],** *The Fractal Geometry of Nature*, WH Freeman.

128.**Mandelbrot, B. B. [1997],** Ed., *Fractals and Scaling In Finance*, Springer-Verlag, ISBN 0-387-98363-5.

129.**Marko, K., James, J., Dosdall, J. and Murphy, J. [1989]**, Automotive Control System Diagnostics Using Neural Nets for Rapid Pattern Classification of Large Sets, *Proc. of Int. Joint Conf. on Neural Networks*, II, 13-15.

130.**Martinetz, T.M., Ritter, H.J. and Schulten, K.J. [1989]**, 3D Neural Net for Learning Visuomotor Coordination of a Robot Arm, *Proc. Int. Joint Conf. on Neural Networks*, II, pp 351-356.

131.**Massopust, P. R. [1995],** *Fractal Functions, Fractal Surfaces, and Wavelets*, Academic press, ISBN 0-12-478840-8

132.**McClellan, J., Schafer, R. and Yoder, M. [1997],** *DSP First: A Multimedia Approach*, Prentice Hall Engineering, Science & Math, ISBN 0-13-243171-8.

133.**McCulloch, W.S. and Pitts, W. [1943]**, A Logical Calculus of the Ideas Imminent in Nervous Activities, *Bull. Math. Bio-Physics*, vol. 5, 115-133.

134.**Miller, W.T. [1987]**, Sensor Based Control of Robotic Manipulators using General Learning Algorithm, *IEEE Journal of Robotics and Automation* 3: 157-165.

135.**Miller, W. T., Glanz, F. H. and Kraft, L. G. [1987]**, Application of a General Learning Algorithm to the Control of Robotic Manipulators, *The Int. J. of Robotic Research,* 6, 84-98.

136.**Miller, W.T., Sutten, R.S. and Werbos, P.J. [1990]**, Eds., *Neural Networks for Control*, London: The MIT Press.

137.**Minsky, M. and Papert, S. [1969]**. *Perceptrons*, Cambridge, MA: MIT Press.

138.**Mitchell, R.J. and Bishop, J.M. [1995]**, Speech, Vision and Colour Applications. In *Neural Network Applications in Control*, Irwin, G.W., Warwick, K., and Hunt, K.J. [Eds.], The Institution of Electrical Engineers.

139.**Molgedey, L., Schuchardt, J. and Schuster, H.G. [1992]**, *Phys. Rev. Lett.* 69, 3717

140.**Moody, J. and Darken, C.J. [1988]**, Learning with Localised Receptive Fields. In *Proceedings of the 1988 Connectionist Model Summer School*, [Eds.], D. Touretzky, G.H. and Sejnowski, T., 133-143. San Mateo, CA: Morgan Kaufmann.

141.**Moody, J. and Darken, C.J. [1989]**, Fast Learning Networks of Locally-Tuned Processing Units, *Neural Computing* 1[2]: 281-294.

142.**Moss, F. and McClintock, P.V.E. [1989]** Eds. *Noise in dynamical systems*, Cambridge University Press, Cambridge, UK vol. 1-3.

143.**Moss, F., Schleich, W. and Lugiato, L. A. [1990]** Eds. *Noiseand Chaos in Nonlinear Dynamical Systems*, Cambridge University Press, Cambridge, UK.

144.**Moss, F. [1993]** Ed., *J. Stat. Phys.,* A Bulsara and M Shlesinger, 70, 1-514.

*145.***Murray-Smith, R., Neumerkel, D. and Sbarbaro-Hofer, D. [1992]**, Neural Networks for Modelling and Control of a Non-linear Dynamic System, *Proc. 7th IEEE Int. Symposium on Intelligent Control.*

146.**Nakajima, M., Sakaue, K. and Asada, M. [1990]**, Image Processing and Computer Vision, *J. Inst. Telev. Eng.,* Japan, 44, 889-894.

147.**Oppenheim, A. V. and Schafer, R. W. [1989],** *Discrete Time Signal Processing*, Prentice-Hall Engineering, Science & Math, ISBN 0-13-216292-X

148.**Oppenheim, A. V. and Schafer, R. W. [1975],** *Digital Signal Processing*, Prentice Hall Engineering, Science & Math, ISBN 0-13-214635-5.

149.**Orfanidis, S. [1995],** *Introduction to Signal Processing*, Prentice Hall Engineering, Science & Math, ISBN 0-13-209172-0.

150.**Ortega, R. and Tang, Y. [1989]**, Robustness of Adaptive Controllers - A survey, *Automatica*, vol. 25, No. 5, 651-677.

151.**Page, G.F., Gomm, J.B. and Williams, D. [1993]**, Eds., *Application of Neural networks to Modelling and Control*, London: Chapman and Hall.

152.**Parzen, E. [1962],** On estimation of a probability density function and mode, *Ann. Math. Statist.* Vol. 33, pp. 1065-1076.

153.**Peitgen, H.O. and Saupe, D. [1988],** Eds., *The Science of Fractal Images,* Springer-Verlag.

154.**Peitgen, H. [1992],** *The Beauty of Fractals, Images of Complex Dynamical Systems*, Springer-Verlag.

155.**Peitgen, H. [1992],** *Chaos and Fractals, New Frontiers of Science*, Springer-Verlag, ISBN 0-387-97903-4.

156.**Peters E E, [1994],** *Fractal Market Analysis: Applying Chaos Theory to Investment and Economics*, Wiley, ISBN 0-471-58524-6.

157.**Poggio, T. and Girosi, F. [1990]**, Networks for Approximation and Learning, *Proc. IEEE*, vol. 78, No. 9, Sep. 1481-1497.

158.**Psaltis, D., Sideris, A. and Yamamura, A.A. [1988]**, A Multi-layered Neural Network Controller, *IEEE Control Systems Magazine*, April, 17-21.

159.[1]**Rabiner, L R. and Schafer, R. W. [1978],** *Digital Processing of Speech Signals*, Prentice Hall Engineering, Science & Math, ISBN 0-13-213603-1

160.**Rabinovich, M. I. [1996],** Chaos and neural dynamics, *Radiophysics and Quantum Electronics,* vol. 39, No. 6.

161.**Rasmus, D.W. [1991],** Expert Systems, *Byte*, 16, 1, 281-285.

162.**Read, R. [1997],** *Essence of Communications*, Prentice Hall Professional Technical Reference, ISBN 0-13-521022-4

163.**Rosenblatt, M. [1956]**, Remarks on some nonparametric estimates of a density function, *Ann. Math. Statist.* Vol. 27, pp. 832-837.

164.**Rosenblatt, F. [1958]**, The Perceptron: A Probabilistic Model for Information Storage and Organisation in the Brain, *Psychological Review*, 65, 386-408.

165.**Rosenblatt, F. [1961]**, *Principles of Neurodynamics*, New York: Spartan.

166.**Rumelhart, D.E. and McClelland, J.A. [1986]**, Explorations in Parallel Distributed Processing: *A Handbook of Models, Programs, and Exercises*, Includes Software. A Bradford Book, The MIT Press.

167.**Rumelhart, D.E., Hinton, G.E. and Williams, R.J. [1986]**, Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing*, vol. 1, Bradford Books - The MIT Press.

168.**Russ, J. C. [1994],** *Fractal Surfaces*, Plenum Press, ISBN 0-306-44702-9.

169.**Saic's manual, [1989]**, *Science Applications International Corporation's Manual for Ansim - An artificial neural systems simulation programme*. Serial #: ANSKPRG 2.3 906, San Diego, CA92121: SAIC.

170.**Sanner, R. M. and Akin, D. L. [1990]**, Neuromorphic Pitch Attitude Regulation of an Underwater Tele-Robot. *IEEE Control Systems Magazine,* 10, 62-67.

171.**Schroeder, M. S. [1991],** *Fractals, chaos, power laws: minutes from an infinite*

paradise, New York: W.H. Freeman

172. **Sejnowski, T. and Rosenberg, C. R. [1986],** A multi-purpose neural processor for machine vision systems, *IEEE Trans. Neural Networks.*

173.**Shenoi, K. [1995],** *Digital Signal Processing In Telecommunications*, Prentice Hall Professional Technical Reference, ISBN 0-13-096751-3.

174.**Shinsky, F.G. [1987]**, *Distillation Control for Productivity and Energy Conservation*, New York: McGraw-Hill.

175.**Sklar, B. [1988],** *Digital Communications: Fundamentals and Applications*, Prentice Hall Professional Technical Reference, ISBN 0-13-211939-0.

176.**Soderstrom, T. and Stoica, P. [1989]**, *System Identification.*, Hemel Hempstead, U.K: Prentice-Hall.

177.**Sofge, D.A. and White, D.A. [1992]**, Applied Learning Optimal Control for Manufacturing, 259-282. In *Handbook of Intelligent Control*, White, D.A. and Sofge, D.A. [Eds.], New York: Van Nostrand Reinhold.

178.**Solo, V. and Kong, X. [1994],** *Adaptive Signal Processing Algorithms: Stability and Performance*, Prentice Hall Engineering, Science & Math, ISBN 0-13-501263-5.

179.**Stinchcome, M. and White, H. [1989]**, Universal Approximation using Feed-forward Networks with Non-Sigmoid Hidden Layer Activation Functions. In *Proc. of the Int. Joint Conf. on Neural Networks* [June], vol. 1, 607-611, Washington DC: IEEE Neural Network Committee.

180.**Stoyan, D. and Stoyan, H. [1994],** *Fractals, Random Shapes and Point Fields: Methods of Geometrical Statistics*, Wiley, ISBN 0-471-93757-6.

181. **Takayasu, H. [1990],** Fractals in the Physical Sciences, *Nonlinear science: theory and applications,* Manchester: Manchester University Press.

182.**Tatom, F. B. [1987],** The Applications of Fractional Calculus to the Simulation of Stochastic Processes, *America Institute of Aeronautics and Astronautic Journal*, Vol. 89, pp. 1-7.

183.**Tricot C. [1995],** *Curves and Fractal Dimension,* New York, London: Springer-Verlag.

184.**Tsaptssinos, D., Jalel, N.A. and Leigh, J.R. [1993]**, Estimation of State Variables of a Fermentation Process via Kalman Filter and Neural Network. In *Application of Neural Networks to Modelling and Control*, Page, J.B., Gomm, J.B. and Williams, D. [Eds.], 53-

73, London: Chapman and Hall.

185.**Ungar, L.H. [1990]**, A Bioreactor Benchmark for Adaptive Network-Based Process Control. In *Neural Networks for Control*, Miller, W.T., Sutton, R.S. and Werbos, P.J. Eds., London: The MIT Press.

186.**Ungar, L.H., Powell, B.A. and Kamens, S. N. [1990]**, Adaptive Networks for Fault Diagnosis and Process Control, *Computers Chem. Eng.*, 14, 561-572.

187.**Vidyasagar, M. [1985]**, *Control Systems Synthesis: A Factorization Approach*, Cambridge, MA: MIT Press.

188.**Voss, R. F. [1979],** 1/f (flicker) noise: a brief review, *Proceedings of the 32nd Annual Symposium on Frequency Control*, Atlantic City, pp 40-46.

189.**Voss, R. F. [1985],** Scaling phenomena in disordered systems, in *Random Fractals: Characterisation and Measuremen*t. New York, Plenum Press, Ed. R Pynn and A Skjeltorps, ISBN 0-306-42112-7.

190.Waibel, A., Hanzawa, G., Hinton, K., and Lang, K.J. [1989], Phoneme recognition using time-delay neural networks, *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, no 3, pp. 328-339.

191.**Wasserman, P. [1989]**, *Neural Computing Theory and Practice*, New York, NY: Von Nostrand Reinhold.

192.**Wasserman, P.D. [1993]**, *Advanced Methods in Neural Computing*, New York: Van Nostrand Reinhold.

193.**Watson, G.A. [1980]**, *Approximation Theory and Numerical Methods*, New York: John Wiley and Sons.

194.**Weeks, E. R. and Burgess, J. M. [1997],** Evolving artificial neural networks to control chaotic systems, in *Physical Review E,* VOLUME 56, Number 2.

195.**Werbos, P.J. [1988]**, Generalisation of Back-propagation with Applications to a Recurrent Gas Market Model, *Neural Networks*, 1: 339-356.

196.**Werbos, P. J. [1990]**, Back-propagation Through Time: What It Does and How To Do It, *Proc. of IEEE,* 78, 1550-1560.

197.**Werbos, P.J., McAvoy, T. and Su, T. [1992]**, Neural Networks, System Identification and Control in the Chemical Process Industries. In *Handbook of intelligent control*, White, D.A. and Sofge, D.A. [Eds.], New York: Van Nostrand Reinhold.